KULKARNI, ANUPAM AJAY. Development of an Eulerian Lagrangian Method to Simulate Dispersed Multiphase Flows with Complex Spray Dynamics. (Under the direction of Jack Edwards).

An Eulerian-Lagrangian framework is developed to allow for the simulation of dispersed multi-phase flow, with an emphasis on capturing complex physics relevant for effectively modeling spray dynamics. The Lagrangian description of the discrete phase comes from the droplet tracking engine, the development of which is the focus of this work. This is coupled with the existing CFD code REACTMB to allow for 3-D RANS or LES flow simulations using multi-block curvilinear grids over a variety of flow regimes. A linear time search algorithm is developed to locate droplets in curvilinear grids. A variety of force models including drag, lift, gravity and buoyancy act on the droplets through interpolated flow variables. Spray dynamics is captured by modeling droplet physics such as droplet breakup and vaporization. Droplet breakup is modeled using the Taylor Analogy Breakup model, which assumes that the behavior an oscillating, distorting droplet due to aerodynamic drag is analogous to a forced harmonic oscillator. Droplet generation after breakup is calculated using the Cascade Analogy Breakup model. Additionally, droplet vaporization is captured using the Langmuir Knudsen-1 model which accounts for non-equilibrium effects in evaporation, while assuming a single droplet temperature. The droplet engine can also handle immersed boundaries for injection and deposition of droplets to allow for unique simulation possibilities. The engine is parallelized using OpenMPI using the same domain decomposition as the CFD code, allowing for transient droplet tracking. The droplet engine has been used to successfully track over 100 million droplets in some cases and routinely tracked $\sim 15$ million droplets without a significant drop in speed. Droplet transport has been simulated for high pressure injection into a quiescent environment and transverse injection into a subsonic cross-flow. The simulated droplet populations generally show good agreement with experimental results; velocity slip between the two phases is naturally captured, droplet response to underlying flow based on its size is physically consistent, and droplet breakup responds to shocks as expected. The droplet breakup and vaporization models have been validated with experimental results. Additionally, contaminant transport modeled as a droplet population has been simulated in closed spaces, using droplets injected from immersed boundaries. Droplets also successfully interact with IBs and currently get deposited onto them. Overall a highly flexible, robust and efficient droplet tracking engine to effectively couple with the REACTMB CFD solver is developed, capable of handling a wide plethora of real world engineering problems. Possible avenues for future extension of this work are dynamic load balancing by accounting for droplet loading to improve efficiency, two-way coupling with the gas phase, modelling droplet
collisions and addition of physics based deposition and re-aerosolization models.
Development of an Eulerian Lagrangian Method to Simulate Dispersed Multiphase Flows with Complex Spray Dynamics

by
Anupam Ajay Kulkarni

A dissertation submitted to the Graduate Faculty of North Carolina State University in partial fulfillment of the requirements for the Degree of Doctor of Philosophy

Aerospace Engineering

Raleigh, North Carolina
2018

APPROVED BY:

Hong Luo
Pramod Subbareddy

Carl Tim Kelley
Jack Edwards
Chair of Advisory Committee
DEDICATION

To my parents.
BIOGRAPHY

The author was born on the 17th of December, 1989 in Jalna, in the state of Maharashtra, India. He received his Bachelor’s degree in Mechanical Engineering from the University of Pune in May 2011. Anupam went on to study Aerospace Engineering in North Carolina State University (NCSU), Raleigh, North Carolina, USA in Fall 2013. Here he began working in the Computational Fluid Dynamics laboratory from January 2014 under the guidance of Dr. Jack Edwards. Overall, Anupam’s interests include physics, applied mathematics and programming.

Aside from work, Anupam loves music, playing the guitar and reading about philosophy and psychology.
ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor Dr. Jack Edwards for his continued guidance. During these last five years, his patience, occasional sharp criticism against my complacency but most of all, his words of encouragement and belief in me and my work have been instrumental in me achieving this milestone. I would also like to thank Dr. Hong Luo for always being approachable and kind in offering help and advice. Similarly, I would like to thank Dr. Tim Kelley and Dr. Pramod Subbareddy for being awesome professors, and for their valuable insight.

The journey to a doctoral degree is a long and difficult one, and I would never have been successful without the support of my family and friends. I would like to thank my parents, Mr. Ajay Kulkarni and Dr. Daya Kulkarni for all their sacrifices to make this opportunity possible and for their love and support throughout the process. Friends have made NC State home away from home and each one of them made this journey memorable; but there are some who deserve a special mention: Gaurav Patwardhan, Nachiket Kulkarni, Aditya Pandare and Nadish Saini who helped me through my personal struggles and motivated me to keep fighting. Finally, I am thankful to all the professors who have taught me in the NC State MAE department, along with support staff who have always been helpful.

The work on this project was supported by the Angel Family Professorship endowment and by other sources. Computer resources were provided by NCSU and by the DoD’s HPCMP program. Special thanks is extended to Dr. Xudong Xiao of Corvid Technologies for his assistance in improving the parallelization strategy.
# TABLE OF CONTENTS

LIST OF TABLES ................................................................. vii

LIST OF FIGURES ............................................................... viii

Chapter 1 Introduction ....................................................... 1

Chapter 2 Droplet Search .................................................... 8
  2.1 Basic Idea ............................................................... 8
  2.2 Algorithm formulation .............................................. 10
  2.3 Implementation Nuances ........................................... 12
    2.3.1 Speed ........................................................... 12
    2.3.2 Stability ...................................................... 13
    2.3.3 Two-step Droplet Search ................................... 13

Chapter 3 Interpolation and Integration .................................. 14
  3.1 Trilinear Interpolation ............................................. 14
  3.2 Force Models ........................................................ 17
  3.3 Numerical Integration .............................................. 18
  3.4 Stochastic Droplet Tracking ..................................... 20

Chapter 4 Droplet Breakup ................................................ 22
  4.1 Predicting a Droplet Breakup Event ............................... 22
    4.1.1 Basic Idea of the Taylor Analogy Breakup Model ........ 22
    4.1.2 Numerical Implementation .................................. 23
    4.1.3 Algorithm .................................................... 26
  4.2 Generation of Product Droplets .................................. 27
    4.2.1 Calculation of the Radius Factor ........................... 27
    4.2.2 Variation of $K_{br}$ with Weber Number .................. 30
    4.2.3 Calculating the Normal Velocity components .......... 32

Chapter 5 Droplet Vaporization .......................................... 34
  5.1 Governing Equations .............................................. 34
  5.2 Implementation Details .......................................... 35
  5.3 Selection of properties .......................................... 37

Chapter 6 Immersed Boundaries ........................................... 39
  6.1 Data Structures for Immersed Boundaries ....................... 39
  6.2 Droplet Interaction with Immersed Boundaries ................. 40
  6.3 Droplet Injection from Immersed Boundaries .................... 42

Chapter 7 Parallelization .................................................. 43
  7.1 Basic Objectives and Challenges ................................ 43
  7.2 Grid Connectivity for Droplet Tracking ...................... 46
Chapter 8 Auxiliary Functions ......................................................... 53
  8.1 General Algorithm and Data Structures ................................. 53
    8.1.1 Algorithm .................................................................. 53
    8.1.2 The state flag ............................................................. 54
  8.2 Handling Boundary Conditions .............................................. 55
    8.2.1 Wall ........................................................................ 56
    8.2.2 Symmetry ................................................................... 56
    8.2.3 Periodic ...................................................................... 56
    8.2.4 Outlet ......................................................................... 57
  8.3 Droplet Injection .................................................................... 57
    8.3.1 Face Based Injection .................................................... 58
    8.3.2 Block Based Injection ................................................... 58

Chapter 9 Results ........................................................................... 60
  9.1 Aerated Liquid Injection into Quiescent Environment ................. 61
  9.2 Droplet Transport in a C-17 .................................................... 65
  9.3 Air Assisted Liquid Drop Atomization .................................... 70
  9.4 Planar aerated liquid jet injection with droplet breakup ............... 75
  9.5 Injection into a Subsonic Cross-flow ...................................... 81
    9.5.1 Results without Droplet Breakup .................................. 83
    9.5.2 Results with Droplet Breakup ....................................... 86
  9.6 Single Droplet Vaporization .................................................. 94

Chapter 10 Conclusions ................................................................. 96
  10.1 Discussion of Current Work ................................................ 96
  10.2 Future Work ...................................................................... 97

References .................................................................................... 99
LIST OF TABLES

Table 2.1 Droplet search: Distribution of coefficients .................................. 11
LIST OF FIGURES

Figure 2.1  Droplet search: Basic idea ............................................ 9
Figure 3.1  Trilinear Interpolation: Mapping to the unit cube .............. 15
Figure 4.1  Radius factor variation with Weber number in ETAB and CAB models ....... 30
Figure 4.2  Radius factor variation with Weber number in the implemented model ...... 32
Figure 6.1  STL Rendering of an Object .......................................... 41
Figure 7.1  Neighbor zones for a block ........................................... 44
Figure 7.2  3 stage linked list for grid connectivity ............................ 47
Figure 9.1  Centerplane snapshot of density contours inside an ‘out-in’ aerated liquid injector .......................................................... 61
Figure 9.2  Vapor-phase mass fraction contours in aerated-liquid jet plume .......... 62
Figure 9.3  Center plane droplet populations contours with 50 $\mu m$ annular droplets (shown in red) and 10 / 20 $\mu m$ core droplets (shown in blue) .................. 63
Figure 9.4  Composite image super-imposing vapor-phase mass fraction contours with a droplet field snapshot ........................................ 64
Figure 9.5  Comparison of predicted droplet fields with experimental shadowgraph results 64
Figure 9.6  Rendered interior of C-17 MedEvac showing mesh slices (see closeup view for detail) ................................................................ 65
Figure 9.7  Aerosolized droplet population at 125 s after sneeze for different patients 67
Figure 9.8  Droplet locations at different times (Case 1) .......................... 68
Figure 9.9  Deposition patterns for different sneezing events after 20 min .......... 68
Figure 9.10 Droplet tracks showing initial entrainment of droplets into low-momentum region in front of wing root cutout ........................................ 69
Figure 9.11 Schematic diagram of the experimental apparatus .......... 70
Figure 9.12 2D velocity profile for $We = 18$ .................................... 71
Figure 9.13 Velocity profile comparison for $U = 59 m/s$ ......................... 71
Figure 9.14 Comparison of Droplet trajectories ................................... 72
Figure 9.15 Comparison of computed Sauter mean diameter distribution with experimental results ................................................................. 73
Figure 9.16 Close up view of the breakup process ............................... 74
Figure 9.17 U-velocity component distribution at $Z = 0$ .................... 75
Figure 9.18 U-velocity component distribution at $Z = 0$ .................... 76
Figure 9.19 Droplet population after 250 timesteps .......................... 77
Figure 9.20 Droplet population after 750 timesteps .......................... 77
Figure 9.21 Droplet population after 1250 timesteps .......................... 78
Figure 9.22 Droplet population after 2000 timesteps .......................... 78
Figure 9.23 Droplet population after 2500 timesteps .......................... 79
Figure 9.24 Droplet breakup due to shock ........................................ 80
Figure 9.25 Liquid density contours: out-in injector (top: instantaneous; bottom: average) 81
Chapter 1

Introduction

A wide variety of real world engineering problems of interest belong to the class of dispersed multi-phase flow, with examples ranging from particulate re-suspension, contaminant transport, and erosion to high speed spray combustion. Naturally, there is a strong motivation to develop techniques to effectively simulate these flows. Problems such as high speed spray combustion are important topics of research, studied both experimentally and computationally. Spray dynamics involve complex physical phenomenon such as droplet breakup and vaporization. Practical applications of sprays also involve the presence of a large number of droplets, subject to a wide variety of forces. In the case of high speed turbulent combustion of liquid fuels, there are a variety of flow structures that droplets interact with and respond to. Successful simulation of spray dynamics is thus a challenging and important problem.

Dispersed multi-phase flows are simulated using two primary classes of methods: Eulerian-Eulerian (E-E), where an Eulerian description of both the continuous and the dispersed phase is used [29], and Eulerian-Lagrangian (E-L) methods, where the continuous phase is still described in an Eulerian paradigm, whereas the discrete phase is tracked using a Lagrangian description. In general, both methods have their advantages and disadvantages; however, they differ significantly in their set of approximations and their implementation. The simplest E-E method for simulating two-phase flow is the mixture model equation system, which assumes no slip between the two phases [2,10]. However, in the case of spray simulations, this tends to be a fairly restrictive assumption. For high speed cases, velocity slip between the two phases can be quite large, and it dictates the general dynamics of the spray through advection, breakup and vaporization. Other E-E methods such as the two-phase, two-fluid models solve continuum equations for both phases separately, and can thus capture velocity slip effects [25]. However, this increases the total degrees of freedom of the problem, consequently increasing computational cost. Additionally, two-fluid equation systems are significantly more difficult to converge and may require special treatment in formulation and numerics to capture higher values of slip.
It is also quite complicated to model other physical phenomenon of interest, such as droplet breakup and vaporization when using E-E methods.

E-L methods treat the droplet as a control mass advecting through the flow field, and thus naturally capture velocity slip between the two phases. The particles or droplets are advected based on the forces acting on them using Newton’s laws of motion, making them relatively easier to implement. Moreover, the governing equations of motions are stable, making them easier to converge, and their numerical implementation is simpler. Since the Lagrangian description of the dispersed flow closely mimics how the discrete phase is observed physically, these methods are more intuitive and easier to interpret. Droplet vaporization and breakup models usually involve integration of additional parameters or droplet properties along the trajectory of the droplet. These are generally simpler to implement in the Lagrangian framework. The disadvantages of E-L methods primarily center around dense spray simulations where the physics becomes more difficult to resolve and coupling with the carrier phase is more challenging.

Lagrangian droplet tracking codes are widely used as a part of various commercial and open source packages. Most of these codes employ Lagrangian droplet tracking as a post processing step. However, to truly resolve spray dynamics, transient droplet tracking is necessary, where both the flow field and droplet population are simultaneously advanced in time. NCSU’s REACTMB CFD solver has been used to simulate a wide range of all speed flow problems, using Reynolds Averaged Navier Stokes (RANS) or Large Eddy Simulation (LES) techniques for turbulence modelling. This solver uses three dimensional, structured curvilinear multi-block grids, is parallelized using OpenMPI and has been tested and validated extensively. A droplet tracking engine was developed from the ground up to work with the existing REACTMB solver to facilitate the best possible coupling between the Eulerian and Lagrangian solvers. This allows the simulation of dispersed multiphase flow using the E-L approach, further extending the capabilities of REACTMB. This thesis discusses the governing equations, numerical methods and practical implementations used in the Lagrangian droplet tracking engine through sequential development and improvements [7, 8] over the course of the last four years.

The droplet engine is designed keeping four key objectives in mind:

1. Accuracy
2. Efficiency
3. Robustness
4. Modularity

Every component of the code follows these objectives in its implementation. The remaining part of this section describes in brief the various aspects of the code, and the motivation behind
the choices made in terms of formulation, model selection and implementation procedure. At its core, Lagrangian droplet tracking advects each droplet in the domain by calculating the forces acting on it based on Newton’s laws. Some Lagrangian tracking codes use the concept of a computational droplet or a parcel, which is an entity which represents multiple physical droplets, sharing similar size, material properties and location. This procedure is primarily used to limit the total number of droplets tracked by the code, thus reducing computational cost. However, this approximation can result in a significant loss of accuracy. Additionally, this idea cannot be used in cases where the underlying flow is resolved with LES. As a result, the developed droplet engine tracks every single droplet in the domain individually for the best possible resolution. Given the scope of realistic spray simulations, this means that the droplet engine must be able to track several millions of droplets.

There are three primary aspects of Lagrangian droplet tracking:

- **Droplet search**
- **Trilinear interpolation**
- **Droplet integration**

3-D curvilinear multi-block meshes are necessary to allow for sufficient flexibility to accommodate the simulation of complex geometries. However, this makes droplet search especially difficult, and possibly computationally expensive. In some cases, this issue is handled by using computational space (C-space) algorithms as opposed to physical space (P-space) algorithms. In C-space algorithms, the 3-D curvilinear grid is mapped to a 3-D Cartesian grid. Once droplet positions in the physical space are also mapped to this computational space, droplet search becomes trivial along with interpolation. However, there are multiple disadvantages to this approach [22, 23]. The mapping functions for such a process would be unique to each cell/node depending on the approach. For transient droplet tracking, C-space algorithms involve a mapping and inverse mapping process for each time-step, making C-space algorithms more expensive than P-space algorithms [23]. Critically, this method makes droplet trajectories susceptible to grid curvature and skewness. Distortions in the grid are necessary to fit meshes to complex geometries, and this makes C-space algorithms less accurate than P-space ones as well [22].

A key aspect of the developed droplet engine is a fast P-space droplet search algorithm which works for 3-D grids in linear time in the worst case. In most iterations the search takes a constant time for completion. The core purpose of droplet search is to determine which cell contains the droplet to calculate the flow values to be used. P-space algorithms for droplet search are iterative algorithms and various options exist [3, 19]. In this work, the method suggested by X. Q. Chen [3] is extended to 3-D grids and further optimized for use in the search algorithm in the engine. This algorithm uses a simple geometric idea to determine if a given coordinate
lies within a given hexahedral cell. This process involves a simple calculation of six face-based coefficients. Additionally, if the coordinate is not present in the cell, the algorithm also gives a simple update along the \(i, j\) and \(k\) directions of the curvilinear grid for the next iteration. Note that this update is simultaneous along each direction since it leverages the \((i, j, k)\) structure of a structured grid, making the search time linear. Since the search operation must be performed for each droplet at every iteration, it must necessarily be fast for an overall efficient algorithm. Details about the exact implementations of droplet search are presented in Chapter 2.

The CFD code solves for flow variables at the cell centers which are then averaged to the nodes. However, since droplet positions do not necessarily coincide with the nodes of the grid, the exact values of flow variables at the droplet location must be determined. To this end, a parametric form of trilinear interpolation is used to map the nodal flow values from the cell containing the droplet to a unit cube. The droplet’s position is then determined in this cube and flow properties at its exact location are calculated. As with droplet search, the droplet is considered to be a point object for the purpose of interpolation. These flow variables are then used to calculate the forces acting on the droplet. Currently, the code has force models for drag [24], lift [6,17], gravity and buoyancy. For stability and robustness, the advection equations are linearized, resulting in a semi-implicit numerical integration scheme. The droplet engine has been tested to be stable for high Mach number flows, where relative velocities can be very high. The detailed formulations for trilinear interpolation, force models and numerical integration are described in Chapter 3.

To effectively simulate sprays, especially in high speed turbulent conditions, it is imperative to model droplet breakup and vaporization processes. Spray simulations in high speed flows undergo primary and secondary breakup. As a result, capturing droplet breakup is key in determining the size of droplets and their distribution. The droplet size distribution is instrumental in determining the overall spray dynamics. Smaller droplets relax to the flow velocity quickly, are more resistant to further breakup, and vaporize faster. Additionally, the vaporization process in smaller droplets is more non-linear. Larger droplets show deeper penetration in cross flow conditions, but have higher Weber numbers and are thus susceptible to breakup. Thus, there is a significant difference in the way droplets respond to the flow based on their size. The droplet size distribution in turn is heavily dependent on breakup and vaporization processes.

The droplet break up model used in the engine is based on the Taylor Analogy Breakup (TAB) [20] model. The TAB droplet breakup model postulates that an an oscillating droplet distorted due Rayleigh-Taylor instabilities is analogous to a forced harmonic oscillator system. The external force is the aerodynamic drag acting on the droplet, the restoring force is the surface tension and the damping force is due to droplet viscosity. A normalized distortion parameter and its distortion rate are numerically integrated over the droplet lifetime, with both governed by the forced harmonic oscillator equation. A breakup event is said to have occurred
when this distortion parameter crosses a certain threshold. The TAB model is known to under-
predict radius of the product droplets [27], which led to the development of the Enhanced Taylor
Analogy Breakup (ETAB) [27] model. This model describes the droplet generation process as a
first order rate process. Subsequent development of the ETAB model resulted in the Cascade
Analogy Breakup (CAB) model [28], capable of handling droplets over a wide range of Weber
numbers. This model accounts for breakup processes in the bag, stripping and catastrophic
droplet breakup regimes, and hence has been implemented in the droplet tracking engine to
model the radius factor after breakup. However, the exact interpretation of the radius formul-
ation as given by the CAB model is unclear. As a result, multiple interpretations of the radius
factor have been described, and results with some of these approaches have been presented.
Note that the decision to break up a droplet is still based on the TAB model. Details regarding
the TAB and CAB models are specified in Chapter 4.

Droplet vaporization is modeled using the formulation given by Miller et. al [18]. Droplet
vaporization is captured by integrating two additional differential equations for droplet mass
flow rate due to evaporation, and droplet temperature. Miller et. al. compare the performance of
seven different evaporation models based on the evolution of droplet temperature and size over
time as predicted by each model. The LK-1 model is based on the Langmuir Knudsen law and
captures non-equilibrium evaporation. The choice of this model is due to its accuracy as docu-
mented by Miller et. al, and its lower computational cost. Due to the emphasis on application
of the droplet engine for high speed turbulent combustion, capturing non equilibrium effects
which are significant in the case of smaller droplets and higher velocity slip is important. Note
that a single droplet temperature is used and thus variation of temperature inside the droplet
is ignored. Details about the formulations and models used to implement droplet vaporization
are given in Chapter 5.

Immersed boundary methods provide a way to simulate quite unique engineering problems;
as an example, contaminant transport in an enclosed space where features and people in the
domain were rendered as immersed boundaries has been simulated with REACTMB and the
developed droplet tracking engine. The REACTMB CFD solver has an in-built capability to
handle immersed boundaries, and this was extended to the droplet engine. Immersed boundary
methods can be used to simplify the rendering of complex objects in the domain. Details about
NCSU’s immersed boundary methods are explained in [4, 5]. Droplets interact with immersed
boundaries and deposit on them. Additionally droplets may be injected from certain demarcated
zones of the immersed boundaries as well. Details about immersed boundaries and droplet
interaction and injection are presented in Chapter 6.

Simulation of current day engineering problems necessitates parallelization of the code to
make it efficient enough to handle industry scale problems. As discussed before, spray simula-
tions involve tracking tens of millions of droplets. Additionally, since parcels are not used and
every droplet must be resolved, parallelization of the code is necessary to make the code cost
effective for simulating practical spray systems. The droplet engine has also been parallelized
using OpenMPI, keeping the same domain decomposition as the REACTMB CFD code. Each
processor can see a part of the domain; a set of blocks of the complete mesh on which it per-
forms the numerical calculations for CFD. Additionally, it maintains a list of droplets which
are present in the domain mapped to said processor.

Each processor needs the domain and flow information corresponding to the droplets it
tracks in the case of transient droplet tracking. During the advection process, droplets may
leave a certain block and flow into another. Since different blocks are mapped to different pro-
cessors, droplet information must thus be exchanged between processors. Due to the Eulerian
description of the CFD solver, the parallelization procedure used and the connectivity informa-
tion required to implement it is relatively simple. For multi-block structured CFD solvers, each
block requires CFD data from adjacent blocks it shares faces with. As a result, the combinations
of processors which exchange CFD data and the amount of information exchanged are constant
over time. However, due to the Lagrangian description of the droplets, data may or may not be
exchanged with all adjacent blocks. There may be be droplets advected out of a certain block
across edges or corners (in addition to faces) into new blocks, and thus additional connectivity
information must be pre-computed. Similarly, the number of droplets being advected from one
block into another is variable; as the underlying flow changes, or with the addition of random
turbulent dispersion, the droplet response also changes. So the communication patterns between
processors are transient over the course of the simulation. These challenges are addressed by
using a new grid connectivity and two-step exchange process as a part of the droplet engine,
the details can be found in Chapter 7.

Apart from the basic components of the code that were just discussed, there are a few
auxiliary aspects like handling boundary conditions and droplet injection which are discussed
in Chapter 8. The droplet engine can handle a variety boundary condition types, such as walls,
symmetry, periodic and outlets. Various droplet injection functions are written and can utilize
pre-specified droplet size distributions for droplet injection, details of which are also stated
in Chapter 8. Finally, results generated using the developed engine have been presented in
Chapter 9 to showcase its functionality in a variety of applications and flow regimes. The
droplet engine has been used to simulate aerated liquid injection, by representing the liquid
mass flow as a droplet population to compare the jet spreading rate with experimental data.
A smaller case with the aerated liquid injection uses the droplet engine with the breakup
model as a heuristic technique to analyze droplet distribution and behavior in supersonic flow.
Contaminant transport through patients sneezing in a C-17 medical evacuation aircraft has
been simulated by rendering people and objects inside the domain as immersed boundaries.
Results have also been presented for aerated liquid injection into subsonic cross-flow with and
without droplet breakup. Air-assisted liquid atomization of a single droplet is simulated and used to validate the droplet breakup model with experimental data. Additionally, the droplet vaporization model has also been used, to simulate droplet size and temperature of an isolated hexane droplet suspended in a constant velocity high temperature air flow.

Put together, a Lagrangian droplet tracking engine was developed from the ground up to completely couple with the existing REACTMB CFD solver. As will be seen in the results presented later, this engine successfully tracked over a 100 million droplets for some cases and routinely tracked 10 to 20 million droplets without significant penalties in efficiency. The core of the work presented in this thesis is an exercise in tool development. As such, it is the first iteration of the droplet engine albeit repeatedly improved, and to this end, efforts have been taken to make the engine as modular in design as possible. This allows future extensions to the code, which can be made to improve existing functionality or possibly add new ones. This will allow REACTMB to be an extremely powerful solver specifically for spray combustion and in general, for dispersed multi-phase flows.
Chapter 2

Droplet Search

Droplet search is the first step of the droplet tracking algorithm. The purpose of search is to determine the cell which contains the droplet, so that the flow variables associated with this cell may then be interpolated to the droplet location. These interpolated flow variables will then be used to calculate the forces acting on the droplet. Since the CFD code uses 3-D curvilinear grids which are required for fitting the mesh to complex geometries, the searching process is quite tricky and non-trivial. Also, since droplet search is an iterative process that must be performed for each droplet in every iteration (or every sub-iteration), having a fast algorithm is necessary for overall efficiency. The iterative search process starts from an initial guess cell which is chosen wisely for better performance. A linear time algorithm which leverages the structured nature of the mesh progressively searches through the grid, sequentially moving toward the actual solution. The process is said to converge when the cell containing the droplet is located. For the sake of droplet search, the droplet is considered to be a point entity. A cell is said to contain a droplet if its center lies anywhere in the interior of the cell or on one of its faces, edges or vertices.

2.1 Basic Idea

The droplet search algorithm must satisfy two primary conditions:

- Determine if the droplet lies in a given cell
- If not, give a new iterate which is closer to the actual solution

The key idea of the search algorithm is to geometrically determine the position of the droplet with respect to the faces of a hexahedral cell. The algorithm looks at each of the six faces of a hexahedral cell to determine if the droplet position is on the same side as the cell center for each face. For any convex hexahedral cell, the cell center must lie in the interior of the cell.
Therefore, if for each of the six faces both the droplet and cell center lie on the same side, we can conclusively say that the droplet lies inside the cell. Thus the condition for finding a droplet in a cell is split into six sub-checks, one for each face. These sub-checks involve the calculation of a coefficient whose value is one of \{-1, 0, 1\}. If the droplet position is on the same side of a face as the cell center the coefficient takes a value of 1; if it is on the opposite side, it takes the value –1. In the case that the droplet lies on the face, the coefficient takes the value 0. These six algebraic conditions can successfully determine if the droplet is contained in the cell under consideration.

The second part of this iterative algorithm is to find an update if the droplet is not contained in the cell currently under consideration. This is done by considering the differences of the coefficients generated for the opposite faces of the cell. The differences also provide a means to check if the droplet is in the cell, as will be explained in a later section. In cases when the droplet is not in the cell, these differences provide updates for the indices of the next cell the search algorithm looks at. As an example consider a general hexahedral cell as shown in Figure 2.1 and let its index be \((i', j', k')\). Looking at the figure, we can see that the droplet lies across the east face, and thus the difference of coefficients of the east and west faces will be 2. Thus, the search algorithm will test the cell with index \((i' + 1, j', k')\) and successfully find the droplet. If there was a difference between another set of opposite faces, that update would have been applied simultaneously. Thus the search algorithm moves on to adjoining cells, sequentially moving to the correct cell which contains the droplet.

**Figure 2.1:** Droplet search: Basic idea

This simultaneous update along all three directions \((i, j, k)\) is possible only because we have
a structured grid. This is leveraged to greatly speed up the search process making a linear time search algorithm possible. Note that the face of a general hexahedral cell may not be planar. Averaged face normals are used as the approximate normals for calculating the coefficients. These average values are pre-computed by the CFD code and used directly for the search algorithm.

2.2 Algorithm formulation

Consider a general hex cell as shown in Figure 2.1. The faces along the $i$ dimension are named east and west, those along the $j$ dimension are named north and south, and those along the $k$ dimension are named top and bottom, with coefficients for each of these faces being $i_e, i_w, i_n, i_s, i_t$ and $i_b$ in that order. Let us consider in detail the calculation of the coefficient for the east face. Let $P_0$ denote the cell center of the cell shown and $P_2$ represent the face center of the east face. Let $X_p$ denote the position of the droplet and $r_p$ be the vector from $X_p$ to point $P_2$ as shown. Let $n$ denote the averaged outward facing normal for the east face. Then,

$$i_e = -sgn(r_p . n)$$  \(2.1\)

The sign function $sgn$ will return a value of 1 if the product is positive; $-1$ if it is negative and 0 if the product is zero. These values will be calculated for each face. The differences of the coefficients of the opposite faces determine if the droplet is contained in the cell. Note that these are also the update parameters for the next iteration in the event the droplet is not found in the cell. These differences are defined as follows:

$$\Delta_i = 0.5(i_e - i_w)$$  \(2.2\)
$$\Delta_j = 0.5(i_n - i_s)$$  \(2.2\)
$$\Delta_k = 0.5(i_t - i_b)$$  \(2.2\)

Note that $\Delta_{i,j,k}$ are integers and will thus take the values of 0 or 1. Thus, in case the values of $\Delta_{i,j,k}$ is 0.5 which is possible if the droplet lies on one of the faces, edges or corners, it will be rounded down to 0. The following table shows the value of the sum of the coefficients and $\Delta_i, \Delta_j$ and $\Delta_k$ for various droplet positions with respect to the cell.

As seen in Table 2.1, the necessary and sufficient condition for successfully searching a droplet in a cell is that all the $\Delta$ parameters are equal to zero. In case the droplet is not found in the cell $(i,j,k)$ under consideration, the update is as follows:
Table 2.1: Droplet search: Distribution of coefficients

<table>
<thead>
<tr>
<th>Location of droplet</th>
<th>Sum of coefficients</th>
<th>$\Delta_i$</th>
<th>$\Delta_j$</th>
<th>$\Delta_k$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Inside the cell</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>On one of the faces</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>On one of the edges</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>On one of the corners</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Outside the cell, across a face</td>
<td>4</td>
<td>0/1/-1</td>
<td>0/1/-1</td>
<td>0/1/-1</td>
</tr>
<tr>
<td>Outside the cell, across an edge</td>
<td>2</td>
<td>0/1/-1</td>
<td>0/1/-1</td>
<td>0/1/-1</td>
</tr>
<tr>
<td>Outside the cell, across a corner</td>
<td>0</td>
<td>0/1/-1</td>
<td>0/1/-1</td>
<td>0/1/-1</td>
</tr>
</tbody>
</table>

$$
i = i + \Delta_i
$$

$$
j = j + \Delta_j$$

$$
k = k + \Delta_k$$

(2.3)

Overall, the droplet search algorithm can be roughly expressed in Algorithm 1.

Algorithm 1 Droplet Search

Initialize $i_g, j_g, k_g$ to cell IDs of previous time step

$N_{max} = \Sigma$ max indices of the block

Initialize counter to 1

while sum of coefficients < 5 do

Calculate $i_e, i_w, i_n, i_s, i_t$ and $i_b$

Calculate $\Delta_i, \Delta_j, \Delta_k$

if $\Delta_{i,j,k}$ all $== 0$ then

exit loop

end if

$i_g = i_g + \Delta_i$

$j_g = j_g + \Delta_j$

$k_g = k_g + \Delta_k$

if $(i_g, j_g, k_g)$ cross bounds then

Correct $(i_g, j_g, k_g)$

end if
if counter > $N_{\text{max}}$ then
    Set $\text{cellids}(6, np) = 0$
    exit loop
end if

counter = counter + 1

end while

2.3 Implementation Nuances

2.3.1 Speed

Droplet search must be performed for every droplet at each iteration, so even a small reduction in cost of this operation can significantly improve efficiency. Since droplet search is an iterative process, a better initial iterate allows for faster convergence. To this end, the index of the cell containing the droplet for the previous time step is stored and then used as the initial iterate for the current time step. Due to CFL restrictions on the CFD code for stability, the droplet reasonably moves across a couple adjoining cells in a given time step. As a result, in most practical scenarios the droplet location is found in a couple of iterations if the previous cell location is used as the initial guess.

When droplets are transferred from one processor to another, the information regarding the local block id of the new block which contains the cell is known. However, in such cases the initial iterate cannot be the index of the cell for the previous time step since it lies on a different block. So, we consider an initial index that is half of $(i_{\text{max}}, j_{\text{max}}, k_{\text{max}})$ for the new block.

In certain cases as mentioned in Chapter 7, droplet search is used to determine the exact block which may contain a droplet from a possible list of blocks. This process necessitates searching through the entire block and thus determining the correct block by a process of elimination. Though rarely called into action, this process is critical for the successful execution of the engine in a multi-block environment. The droplet search algorithm has an escape ($N_{\text{max}}$ as shown in Algorithm 1) after a certain number of iterations. The point of interest is that this number scales with the sum of block dimensions, rather than the total number of cells in the block. Additionally, the search algorithm also keeps track of its recent search history. If an iterate is repeated it understands that the search process may have encountered a saddle point or that the given block is does not contain the droplet in question. In such circumstances the loop exits immediately, thus saving time.
2.3.2 Stability

To improve stability, the droplet engine has a sub-cycling loop, and the number of sub-cycles can be provided as an input parameter. Sub-cycling essentially cuts the time step by a factor of the number of sub-cycles and is necessary in cases with high speed flows. In cases where droplets cross block interfaces, sub-cycling helps keep the displacement of the droplets within two adjoining cells. This is necessary so that the exact ghost cell containing the droplet is found and the correct boundary condition applied. If a droplet jumps across too many cells close to the interface, a droplet may be lost which makes sub-cycling necessary. However sub-cycling increases the cost per iteration so an optimum value is chosen based on the case being run.

The droplet search algorithm can be broken by using highly non-convex block geometries. These would create saddle points where the search algorithm would essentially get stuck in a loop stuck between two cells. The iterative process has an escape when such a situation arises, followed by a warning message. The search algorithm and the engine overall in general is designed to continue functioning in case the code breaks for a single droplet. In such an event the code will essentially report an error, along with the relevant droplet information. The code has been extensively tested on a variety of grids split across multiple processors, so failure cases are not encountered practically. However, knowing the limitations of the search algorithm is necessary for effective use of the droplet tracking engine.

2.3.3 Two-step Droplet Search

Droplets close to the block boundaries, specifically those within two cell widths of the boundary are searched for a second time after integration. This allows the search algorithm to find droplets in ghost cells. This step is necessary to apply the correct boundary conditions to droplets leaving a block. It is also necessary to determine if the droplet crosses across block interfaces and is tagged for exchange at the end of the current time step.

Droplet search thus has a bounds parameter which limit the domain of search of a block. In the core algorithm, these bounds represent the real boundaries of the block. In the second case they allow for search in ghost cells. Note that this additional search operation does not add to the cost, since if the droplet is inside the domain, its new cell identification numbers are stored. Hence, for the next time step the droplet search returns immediately in one iteration since the cell containing the droplet is already known.
Chapter 3

Interpolation and Integration

3.1 Trilinear Interpolation

Droplet search locates the cell which contains the droplet. After this step the values of flow variables at the exact location of the droplet are calculated. The flow variables from the CFD code are cell centered values, which are averaged to determine the nodal values for all flow variables. These nodal values can then be used for interpolation. Calculating the nodal values of all flow variables is done separately before the start of the droplet tracking engine, at every iteration. Note that these values remain the same through sub-cycling because the CFD solution is considered to be quasi-steady and does not change across sub-cycles.

Trilinear interpolation can only be used on cubes. However the typical cell resulting from grid generation in curvilinear coordinates is a warped hexahedron. As a result each cell containing a droplet at which the function value is to be interpolated must first be transformed to a unit cube. This is best accomplished by using the same isoparametric form to the coordinates of the hexahedron as is used for the interpolation given below.

\[ \phi = a_{\phi,1} + a_{\phi,2}\epsilon + a_{\phi,3}\eta + a_{\phi,4}\zeta + a_{\phi,5}\epsilon\eta + a_{\phi,6}\epsilon\zeta + a_{\phi,7}\eta\zeta + a_{\phi,8}\epsilon\eta\zeta \] (3.1)

Note that \((\epsilon, \eta, \zeta)\) are the coordinates of the droplet in the unit cube which must first be calculated. \(\phi\) is the scalar being interpolated and \(a_{\phi,i}\) are the coefficients which are unique to \(\phi\) and the cell. The \(a_{\phi,i}\) are calculated for each scalar to be interpolated by using the eight known values of the same scalar at each of the eight nodes of the unit cube. These constant values can be algebraically determined for each \(\phi\). Since we map the hexahedral cell to a unit cube, we get:

\[ 0 \leq (\epsilon, \eta, \zeta) \leq 1 \]
Note that in Equation 3.2 the index $i$ in $a_{\phi,i}$ refers to the eight coefficients of the isoparametric form, while the index $j$ in $\phi_j$ denotes the values of $\phi$ at the eight nodes of the unit cube. The mapping of the hexahedral cell to the unit cube, the identifiers of each of the eight nodes and the values of $(\epsilon, \eta, \zeta)$ for the corresponding nodes have been illustrated in Figure 3.1. Since the values of $(\epsilon, \eta, \zeta)$ at the nodes are either 0 or 1 and $\phi_j$ for each of the eight nodes are known, relations between $a_{\phi,i}$ can be easily determined. For example, for node 1, $(\epsilon, \eta, \zeta) = (1, 0, 0)$ and thus $a_{\phi,1} = \phi_1$. We thus have a system of eight equations and eight unknowns for which explicit solutions for each of the $a_{\phi,i}$ can be given as follows:

$$
\begin{align*}
    a_{\phi,1} &= \phi_1 \\
    a_{\phi,2} &= \phi_2 - \phi_1 \\
    a_{\phi,3} &= \phi_4 - \phi_1 \\
    a_{\phi,4} &= \phi_5 - \phi_1 \\
    a_{\phi,5} &= \phi_1 - \phi_2 + \phi_3 - \phi_4 \\
    a_{\phi,6} &= \phi_1 - \phi_2 - \phi_5 + \phi_6 \\
    a_{\phi,7} &= \phi_1 - \phi_4 + \phi_5 + \phi_8 \\
    a_{\phi,8} &= -\phi_1 + \phi_2 - \phi_3 + \phi_4 + \phi_5 - \phi_6 + \phi_7 - \phi_8
\end{align*}
$$

(3.2)
To get the values of flow variables at the droplet location, the value of \((\epsilon, \zeta, \eta)\) must yet be determined. This is done by setting \(\phi\) to the \(x, y\) and \(z\) coordinates of the eight nodes. Using Equations 3.2 we will get the values for \(a_{x,i}\), \(a_{y,i}\) and \(a_{z,i}\). Substituting these values into 3.1 we can write:

\[
X = G(\epsilon)  \tag{3.3}
\]

where \(X\) denotes the position vector of the droplet, which is known. Also \(\epsilon\) denotes the position vector in the unit cube space given by \((\epsilon, \eta, \zeta)\). The resulting equation is then solved for \(\epsilon\). Let,

\[
F(X, \epsilon) = G(\epsilon) - X = 0
\]

Using Newton’s method, we get:

\[
\epsilon^{n+1} = \epsilon^n - [F(X, \epsilon^n)]^{-1} F(X, \epsilon^n)
\]

where,

\[
F\epsilon = \frac{\partial F_i}{\partial \epsilon_j} = M_{ij}  \tag{3.4}
\]

and

\[
M_{ij} = \begin{bmatrix}
a_{x,2} + a_{x,5} \eta + a_{x,6} \zeta + a_{x,8} \eta \zeta & a_{x,3} + a_{x,5} \epsilon + a_{x,7} \zeta + a_{x,8} \epsilon \zeta & a_{x,4} + a_{x,6} \epsilon + a_{x,7} \epsilon \zeta + a_{x,8} \epsilon \eta \\
a_{y,2} + a_{y,5} \eta + a_{y,6} \zeta + a_{y,8} \eta \zeta & a_{y,3} + a_{y,5} \epsilon + a_{y,7} \zeta + a_{y,8} \epsilon \zeta & a_{y,4} + a_{y,6} \epsilon + a_{y,7} \epsilon \zeta + a_{y,8} \epsilon \eta \\
a_{z,2} + a_{z,5} \eta + a_{z,6} \zeta + a_{z,8} \eta \zeta & a_{z,3} + a_{z,5} \epsilon + a_{z,7} \zeta + a_{z,8} \epsilon \zeta & a_{z,4} + a_{z,6} \epsilon + a_{z,7} \epsilon \zeta + a_{z,8} \epsilon \eta
\end{bmatrix}  \tag{3.5}
\]

\(M_{ij}\) is the Jacobian of the isoparametric transformation. Hence \(M_{ij}^{-1}\) must exist so long as the mapping is one to one. Since \(M_{ij}\) is a \(3 \times 3\) matrix we can directly find the inverse as follows:

\[
M^{-1} = \frac{1}{|M|} \begin{bmatrix}
M_{22}M_{33} - M_{23}M_{32} & M_{13}M_{33} - M_{12}M_{33} & M_{12}M_{23} - M_{13}M_{22} \\
M_{23}M_{31} - M_{21}M_{33} & M_{11}M_{33} - M_{13}M_{31} & M_{13}M_{21} - M_{11}M_{23} \\
M_{21}M_{32} - M_{22}M_{31} & M_{12}M_{31} - M_{11}M_{32} & M_{11}M_{22} - M_{12}M_{21}
\end{bmatrix}  \tag{3.6}
\]

Using an initial value of \(\epsilon = (0.5, 0.5, 0.5)\) the iteration converges to an RMS residual of \(10^{-4}\) in about 5 steps.
3.2 Force Models

The interpolated flow variables are then used to determine forces acting on the droplets. Note that the droplet is considered to be a point mass while considering the effects of these forces. The forces used for the development of this engine are drag \([24]\), lift \([6]\), gravity and buoyancy. The governing equations of motion for the droplet may be represented as follows:

\[
\begin{align*}
\dot{m}_p \frac{d\mathbf{v}_p}{dt} &= \sum \mathbf{F} \\
\dot{\mathbf{x}}_p &= \mathbf{v}_p \\
\sum \mathbf{F} &= \mathbf{F}_g + \mathbf{F}_d + \mathbf{F}_l \tag{3.7}
\end{align*}
\]

Note that the subscript \(p\) stands for the droplet (or particle) properties. The properties of the continuous phase are represented by the subscript \(f\). The gravity and buoyancy forces can together be expressed as

\[
\mathbf{F}_g = (\rho_p - \rho_f) \Omega_p \mathbf{g} \tag{3.9}
\]

\[
\begin{align*}
m_p &= \rho_p \left( \frac{\pi d_p^3}{6} \right) = \rho_p \Omega_p \\
\end{align*}
\]

\[
(3.10)
\]

Note that \(\Omega_p\) and \(m_p\) represent the volume and the mass of the droplet respectively. The drag force is formulated based on the droplet Reynolds number \((Re_p)\) based on the relative velocity between the droplet and the continuous phase. This allows for the model to capture the effects of drag across a wide range of Reynolds numbers by varying the drag coefficient \(C_d\).

\[
\mathbf{F}_d = \frac{\rho_f m_p C_d \mathbf{v}_r |\mathbf{v}_r|}{4 \mu_f d_p} \tag{3.11}
\]

The drag coefficient is given by a modification to the Schiller-Naumann \([24]\) drag law:

\[
C_d = \frac{24}{\min(1000, Re_p)} \left( 1 + 0.15 \min(1000, Re_p)^{0.687} \right) \tag{3.12}
\]

\[
Re_p = \frac{\rho_f d_p |\mathbf{v}_r|}{\mu_f} \tag{3.13}
\]

\[
\mathbf{v}_r = \mathbf{v}_r - \mathbf{v}_p \tag{3.14}
\]

Here, \(\mathbf{v}_r\) represents the relative velocity between the two phases.
The two phase version of the REACT-MB code solves for the mixture velocity which is given by:

\[ \mathbf{v}_m = Y_f \mathbf{v}_f + Y_p \mathbf{v}_p; \quad Y_f + Y_p = 1 \] (3.15)

where \( Y_f \) and \( Y_p \) are mass fractions of the continuous and dispersed phase respectively. As a result, we can rearrange the terms of Equation 3.15 to re-define the relative velocity as

\[ \mathbf{v}_r = \frac{1}{Y_f} (\mathbf{v}_m - \mathbf{v}_p) \] (3.16)

The lift forces acting on the discrete phase are calculated based on the model proposed by Lun and Liu [17].

\[ \mathbf{F}_l = \frac{1}{2} \rho_f |\mathbf{v}_r|^2 \frac{\pi d_p^2}{4} \mathbf{C}_L \mathbf{\Omega} \times \mathbf{v}_r \] (3.17)

Here, \( \mathbf{C}_L \) is the lift coefficient given by

\[ \mathbf{C}_L = \frac{d_p |\mathbf{\Omega}|}{|\mathbf{v}_r|} \left( 0.178 + 0.822 Re_p^{-0.522} \right) \] (3.18)

### 3.3 Numerical Integration

The numerical integration of the displacement of the droplet is performed by considering forces per unit droplet mass. We can thus reformulate equations 3.7 and 3.8 to write

\[ \frac{d\mathbf{v}_p}{dt} = \frac{1}{m_p} (\mathbf{F}_g + \mathbf{F}_d + \mathbf{F}_l) \]

\[ \therefore \frac{d\mathbf{v}_p}{dt} = \mathbf{f}_g + \mathbf{f}_d + \mathbf{f}_l = \mathbf{f}_{\text{net}} \]

Here, \( \mathbf{f}_i \) represent the force per unit mass corresponding to \( \mathbf{F}_i \). The discretized form of the above equation is linearized with respect to \( \mathbf{v}_p \) to derive a semi-implicit form of the advection equation. Note that the continuous phase velocity \( \mathbf{v}_f \) from time step \( n + 1 \) is used.

\[ \frac{v_p^{n+1} - v_p^n}{\Delta t} = f_n(v_p^{n+1}, v_p) \] (3.19)
Using a Taylor series expansion,

\[ f_n(v_{p}^{n+1}, v_p) = f_n(v_p^n, v_p) + \frac{\partial f_n}{\partial v_p} (v_{p}^{n+1} - v_p^n) \]

\[ \therefore \frac{v_{p}^{n+1} - v_p^n}{\Delta t} = f_n(v_p^n, v_p) \]

\[ \therefore \left( \frac{1}{\Delta t} - \frac{\partial f_d}{\partial v_p} - \frac{\partial f_l}{\partial v_p} \right) (v_{p}^{n+1} - v_p^n) = f_n(v_p^n, v_p) \]

\[ \therefore v_{p}^{n+1} = v_p^n + \left( \frac{1}{\Delta t} - \frac{\partial f_d}{\partial v_p} - \frac{\partial f_l}{\partial v_p} \right)^{-1} f_n(v_p^n, v_p) \] (3.20)

The Jacobian of the drag term with respect to the droplet velocity is a diagonal matrix with a common entry along the diagonal. This can be calculated using the drag force formulation given in Equation 3.11 and the definition of \( v_r \) as given in Equation 3.16

\[ \frac{\partial f_d}{\partial v_p} = -\frac{3\rho_f C_d |v_r|}{4\rho_p d_p Y_f} \] (3.21)

The Jacobian of the lift forces is a symmetric matrix due to the presence of vorticity \( \Omega \) and thus results in a coupled 3 \( \times \) 3 system. Using the definition of the lift force in Equation 3.17 and the lift coefficient as stated in Equation 3.18 we can write \( F_l \) as follows:

\[ F_l = \frac{1}{2} \rho_f |v_r|^2 \frac{\pi d_p^2}{4} \frac{\Omega \times v_r}{|v_r|} \left\{ \frac{d_p |\Omega|}{|v_r|} \left( 0.178 + 0.822 Re_p^{-0.522} \right) \right\} \]

\[ = \frac{6\rho_f}{8\rho_l} \left( \frac{\pi d_p^3}{6} \right) \left( 0.178 + 0.822 Re_p^{-0.522} \right) \{\Omega \times v_r\} \]

Using the definition of \( m_p \) and \( f_l = F_l/m_p \),

\[ f_l = \phi \{\Omega \times v_r\} \] (3.22)

where,

\[ \phi = \frac{6\rho_f}{8\rho_l} \left( 0.178 + 0.822 Re_p^{-0.522} \right) \]
\[ \frac{\partial f}{\partial \mathbf{v}_p} = \begin{bmatrix} 0 & -\phi \Omega_3 & \phi \Omega_2 \\ -\phi \Omega_3 & 0 & -\phi \Omega_1 \\ \phi \Omega_2 & -\phi \Omega_1 & 0 \end{bmatrix} \tag{3.23} \]

Note that in the calculation of the lift forces, the vorticity \( \Omega \) is determined from the mixture velocity field. This is so, since it is not possible to calculate the vorticity of the vapor-field without the knowledge of the vorticity of the liquid velocity field. As a result, the relative velocity definition used to calculate lift is as given by Equation 3.14.

Equation 3.20 is a \( 3 \times 3 \) system which is solved directly using Cramer’s Rule. The semi-implicit formulation makes the integration stable enough to handle high Mach number flows and high values of relative velocities. The displacement of the droplet is given by the second part of Equation 3.7 and can be discretized as:

\[ x_p^{n+1} = x_p^n + v_p^n \Delta t + 0.5a_p^n \Delta t^2 \tag{3.24} \]

where,

\[ a_p^n = \frac{v_p^{n+1} - v_p^n}{\Delta t} \]

### 3.4 Stochastic Droplet Tracking

Turbulence in flow fields must be accounted for in droplet tracking. In the course of the work presented in this thesis for cases using LES simulations, sub-grid scale velocities were ignored for the purpose of droplet tracking. This was done since the length scale of the Eulerian grid required for LES is very similar to that of the droplets being simulated.

In case of RANS simulations however, the dispersion of particles in the turbulent flow field is accounted by using a stochastic approach. There are various ways to account for turbulence in flow fields and its effects on droplet tracking. In this work, the dispersion of particles in the turbulent flow field is accounted by using a Discrete Random Walk (DRW) approach [30]. The fluctuations in instantaneous velocity due to turbulence are determined by sampling values from a Gaussian distribution. The velocity which the droplet encounters can be split into two components. One of the components is the mean flow velocity which is obtained from the CFD solver and interpolated to the droplet location. The second component is the fluctuating component which is determined from the turbulent kinetic energy (TKE) as follows:

\[ v' = \zeta \sqrt{\frac{2k}{3}} \tag{3.25} \]
In Equation 3.25, $\zeta$ is a vector of three random numbers sampled from a normal distribution, $k$ in the interpolated TKE at the droplet location. Note that each component of $\zeta$ is sampled separately. If the interpolated continuous phase velocity is represented as $v_I$, the actual continuous phase velocity used in the previous calculations will be modified as follows:

$$v_f = v_I + v'$$

(3.26)
Chapter 4

Droplet Breakup

Droplet breakup is observed for a wide range of dispersed multiphase flows and plays a critical role in dictating the droplet distribution. Larger droplets, also referred to as the parent droplets, break into smaller product (or child) droplets under the effect of aerodynamic drag. A droplet’s response to its surroundings is strongly related to the droplet size, with smaller droplets relaxing to the underlying flow field much quicker than large droplets. This makes droplet breakup an important phenomenon in determining overall spray dynamics. Droplet breakup is modelled using the Taylor Analogy Breakup model [20], which models droplet oscillations due to drag as a forced harmonic oscillator. Note that while this model dictates the occurrence of a breakup event, the radius factor and hence the number of generated product droplets is determined by another model. The Enhanced Taylor Analogy breakup (ETAB) model [27], Cascade Analogy Breakup (CAB) model [28] and Modified Cascade Analogy Breakup (MCAB) model [9] are a sequence of later developments that model the generation of product droplets as a first order rate process. The CAB model, interpreted as discussed later, has been implemented into the droplet tracking engine. The governing equations and methodologies for these models are discussed in detail in this chapter.

4.1 Predicting a Droplet Breakup Event

4.1.1 Basic Idea of the Taylor Analogy Breakup Model

The Taylor Analogy Breakup (TAB) model attempts to predict droplet aerodynamic breakup of an oscillating droplet based on an analogy between an oscillating and distorting droplet and a spring mass system [20]. In general, a forced harmonic oscillator has three components - a distorting external force which is independent of the displacement, a restorative spring force proportional to the displacement of the mass and a damping force proportional to the velocity of the moving mass. For a droplet moving in a surrounding gas phase, we have:
• Aerodynamic drag, analogous to the external distorting force
• Surface tension, analogous to the restoring force
• Viscous forces, analogous to the damping force

A key idea of the TAB model is that droplet breakup is governed not just by the Weber number experienced by the droplet at a certain timestep but also by the droplet history. As the droplet follows its streamline, the droplet continues to distort based on the balance of the forces acting on it. This history of relative velocities which the droplet encounters is what determines the breakup rather than a unique critical Weber number. The Weber number is defined as:

$$We = \frac{\rho g |v_r|^2 r}{\sigma}$$  \hspace{1cm} (4.1)

Here, $r$ denotes the droplet radius, $\sigma$ represents the surface tension coefficient and $v_r$ represents the relative velocity between both phases. Note that in theory there may be multiple oscillatory modes for any given droplet. A natural limitation of this model is that only the fundamental mode is tracked, corresponding to the lowest order spherical mode harmonic whose axis is aligned with the relative velocity vector between the droplet and the gas.

The equation of a damped, forced harmonic oscillator is

$$m \ddot{x} = F - kx - d \dot{x}$$  \hspace{1cm} (4.2)

The TAB model involves solving for a normalized distortion parameter $y = x/(0.5r)$. We reformulate Equation 4.2 with the drag force for $F$, the restoring force due to surface tension in $kx$ and viscous damping for $d \dot{x}$ to get an equation in terms of $y$.

$$\ddot{y} = 2\rho g |v_r|^2 - \frac{8\sigma}{\rho r^3} y - \frac{5\mu l}{\rho r^2} \dot{y}$$  \hspace{1cm} (4.3)

This equation is then integrated numerically throughout the droplet lifetime to calculate its distortion. A droplet is said to have broken up once $y > 1$.

### 4.1.2 Numerical Implementation

During the lifetime of the droplet, Equation 4.3 is solved to determine its distortion $y$ and the distortion rate $\dot{y}$. For a constant Weber number, we can get the exact solution for Equation 4.3. In practice, as the droplet is advected, the Weber number does not stay constant along its path. However a quasi-steady state assumption is made to integrate $y$ and $\dot{y}$ numerically by
assuming constant Weber number for a time-step.

\[
y^{n+1} = \frac{We}{12} + e^{-\frac{\Delta t}{td}} \left\{ \left( y^n - \frac{We}{12} \right) \cos(\omega \Delta t) + \frac{1}{\omega} \left( \dot{y}^n + \frac{y^n - We}{td} \right) \sin(\omega \Delta t) \right\} + e^{-\Delta t} \left\{ \frac{1}{\omega} \left( \dot{y}^n + \frac{y^n - We}{td} \right) \cos(\omega \Delta t) - \left( y^n - \frac{We}{12} \right) \sin(\omega \Delta t) \right\}
\]

where,

\[
\frac{1}{td} = \frac{5 \mu_l}{2 \rho_l r^2}
\]

\[
\omega^2 = \frac{8 \sigma}{\rho_l r^3} - \frac{1}{t_d^2}
\]

The material properties of the liquid (dispersed) phase and the gaseous (continuous) phase are represented by subscripts \(l\) and \(g\) respectively. For every timestep and for each droplet the values of \(We\), \(td\) and \(\omega^2\) are first calculated. If the value of \(\omega^2\) is less than zero, that droplet is incapable of breakup. This condition occurs in extremely small droplets and if so we set:

\[
y^{n+1} = 0 ; \quad \dot{y}^{n+1} = 0
\]

In the code these droplets are tagged separately as well, so that they are no longer considered for breakup thus saving computational time. If \(\omega^2 > 0\) then that implies that that droplet can be broken up and further conditions are checked to see if breakup takes place in that particular timestep. The next step involves the calculation of the amplitude of the undamped oscillation, \(A\).

\[
A^2 = \left( y^n - \frac{We}{12} \right)^2 + \left( \frac{\dot{y}^n}{\omega} \right)^2
\]

If \(We/12 + A \leq 1.0\) then the value of \(y\) can never exceed unity, and thus a breakup is impossible in this timestep. In such cases, we simply update \(y\) and \(\dot{y}\) according to Equation 4.4. However, if \(We/12 + A > 1.0\), droplet breakup is possible and we calculate \(t_{bu}\) which denotes the breakup time. We then calculate breakup time assuming that droplet oscillation is undamped for the first oscillation, which will be true for all droplets except very small ones. The time \(t_{bu}\) is the smallest root greater than \(t^n\) of the equation:

\[
\frac{We}{12} + A \cos[\omega(t - t^n) + \phi] = 1
\]
where,

\[ \cos \phi = \frac{y^n - \frac{W_c}{12}}{A} \]

\[ \sin \phi = -\frac{\dot{y}^n}{A\omega} \]

Rearranging the terms in Equation 4.6, we can write

\[ t_{bu} = t_n + \frac{1}{\omega} \left[ \cos^{-1} \left\{ \frac{1}{A} \left( 1 - \frac{W_c}{12} \right) \right\} - \phi \right] \]

\[ = t_n + \frac{1}{\omega} \left[ \cos^{-1} \alpha - \phi \right] \]

In the above equation, a unique solution for \( \phi \) can be obtained since its exact quadrant is known. However, there are infinitely many possible solutions for \( \cos^{-1} \alpha \). The solution to be chosen must be one that gives the minimum value of \( t_{bu} \). However it is not clear how a deterministic solution for the minimum \( t_{bu} \) may be calculated. As a result, three possible roots for \( t_{bu} \) are calculated. The minimum value of \( t_{bu} \) which satisfies \( t_{bu} - t_n > 0 \) is then chosen.

If \( t_{bu} > t^{n+1} \) where \( t^{n+1} = t^n + \Delta t \), then droplet breakup does not occur in the current timestep. In this case \( y \) and \( \dot{y} \) are updated using Equation 4.4. If \( t_{bu} < t^{n+1} \), the droplet under consideration will breakup in the current timestep. Note that the exact time when the condition \( y > 1 \) is satisfied is very close to \( t_{bu} \), though not precisely equal. The \( t_{bu} \) formulation is the solution to \( y = 1 \) assuming a single undamped oscillation of the droplet based on a starting value of \( y \) and \( \dot{y} \) as \( y^n \) and \( \dot{y}^{n+1} \) respectively.
4.1.3 Algorithm

**Algorithm 2 TAB Droplet Breakup**

```plaintext
while $t < t_{\text{max}}$ do
  Calculate $W_c, t_d, \omega^2$
  if $\omega^2 > 0$ then
    Calculate $A$
    if $W_c/12 + A \leq 1$ then
      Calculate $t_{bu}$
      if $t_{bu} < t^{n+1}$ then
        Tag droplet for breakup
        Calculate radius factor
      else
        Update $y$ and $\dot{y}$
      end if
    else
      Update $y$ and $\dot{y}$
    end if
  else
    Update $y$ and $\dot{y}$
  end if
end while
```

Reallocation of new space and copying old data are computationally expensive, so repeating these steps for each breaking droplet would cause the code to slow down significantly. For the sake of efficiency, all new product droplets for a given iteration are all created simultaneously. Droplets breaking in the given timestep are all tagged separately. After the code loops over all existing droplets on a given processor for a timestep, all such droplets undergoing breakup are tagged. The number of product droplets corresponding to each parent droplet is noted. All of the product droplets for that timestep are then generated by creating new entries in the relevant droplet arrays. Each product droplet is assigned the locations of the corresponding parent droplet. The velocity components of the new droplets are copied from their parent droplets and an additional normal component is added to them. The calculation of this normal velocity component will be discussed later.
4.2 Generation of Product Droplets

After a breakup event is determined by the TAB model, the radius factors of the product droplets must be determined. The TAB model itself defines product droplets by calculating a Sauter mean diameter and then sampling product droplet radii from a chi-squared distribution. However, the TAB model greatly under-predicts product droplet radii, thereby generating too many small droplets and overpredicting the degree of the overall breakup process. To address this problem, an alternative definition of the radius factor was proposed by the ETAB model [27], which proposed to model droplet generation as a first order rate process. This model was then improved upon with the CAB model [28], providing radius factor descriptions for a higher range of Weber numbers. The code has the option to calculate radius factors by each of these three methods - however all presented results use the CAB model. Finally, the MCAB model proposes further changes to the radius factor definition based on the injection Weber number. However this model has not been used and thus will not be discussed in this section. Note that each subsequent model proposed includes various modifications and parameter tuning to better fit certain experimental data. These ideas have not been used in the implementation described in this thesis.

4.2.1 Calculation of the Radius Factor

The core idea of the CAB model was introduced by the same author in a previous paper delineating the details of the ETAB model. The main assumption in the ETAB model is that the rate of product droplet generation is proportional to the number of product droplets [27]. The proportionality constant $K_{br}$ depends on the breakup regime. Thus, we can write

$$\frac{dn(t)}{dt} = 3K_{br}n(t)$$  \hspace{1cm} (4.7)

The factor of 3 has been included to simplify later expressions. Using the conservation of mass principle, we can express $n$ as $m_0/\bar{m}(t)$ where $m_0$ is the mass of the parent droplet and $\bar{m}$ is the average mass of the product droplets. Thus, in terms of mass we can state

$$\frac{d\bar{m}}{dt} = -3K_{br}\bar{m}$$  \hspace{1cm} (4.8)

If we assume that the product droplets are generated as a uniform distribution, the above expression can be simplified further. Let $r$ and $a$ denote the radii of the product and parent droplets respectively. We can thus replace $\bar{m}$ simply with $m$ since the average product droplet
mass is equal to the mass of each product droplet.

\[ m = \rho_l 4\pi r_p^3 \]

\[ \therefore \frac{dm}{dt} = \rho_l 4\pi r_p^2 \frac{dr_p}{dt} = -3K_{br} \frac{\rho_l 4\pi r_p^3}{3} \]

\[ \therefore \frac{dr_p}{dt} = -K_{br} r_p \]

\[ \therefore \frac{dr_p}{r_p} = -K_{br} dt \]

Integrating, we get

\[ \ln \left( \frac{r}{a} \right) = -\int_{t_0}^{t_{bu}} K_{br} (We(t)) dt \]

\[ \therefore \frac{r}{a} = \exp \left( -\int_{t_0}^{t_{bu}} K_{br} (We(t)) dt \right) \] (4.9)

Since the mass (and thus volume) of the parent droplet is conserved, the number of product droplets generated \((N_{br})\) by a single parent droplet if the radius factor is \(r/a\) can be expressed as:

\[ N_{br} = \left( \frac{a}{r} \right)^3 \] (4.10)

The ETAB model [27] specifies the expression for the radius factor \(r/a\) as given by Equation 4.11.

\[ \frac{r}{a} = e^{-K_{br} t_{bu,r}} \] (4.11)

Thus, there is a mathematical discrepancy between the two expressions for the radius factor; Equation 4.9 is in fact a more generalised form of Equation 4.11. As a result, the exact interpretation of the expression for the radius factor is unclear. Note that in the ETAB paper [27], the exponent term is stated as \(-K_{br} t_{bu}\); it has been replaced by \(-K_{br} t_{bu,r}\) in Equation 4.11 for notational convenience. This allows for the possibility that \(t_{bu}\), as calculated by the TAB model to determine the timestep of droplet breakup may be different from \(t_{bu,r}\), which is used to calculate the radius factor. The ETAB paper simply states that \(t_{bu,r}\) is calculated
using Equation 4.6 from the TAB model but does not explicitly state equivalence between the two \( t_{bu} \) definitions. As a result, three possible interpretations of the radius factor formula were considered.

Over the course of a droplet lifetime, the Weber number is usually never constant. However, as seen from the two equations for the radius factor, Equation 4.9 reduces to Equation 4.11 if \( K_{br} \) is assumed to be constant over the droplet lifetime. As a result, \( K_{br} \) was calculated at the Weber number value during breakup and this value was assumed to be frozen over the droplet lifetime. To implement this approach, we directly use Equation 4.11, with \( t_{ba,r} = t_{ba} \).

In the second approach, Equation 4.9 was numerically integrated as:

\[
\therefore \frac{r}{a} = \exp \left( -\sum_{i=0}^{n+1} K_{br} (We(t)) \Delta t \right)
\] (4.12)

Note that in Equation 4.12, for the final time step when breakup occurs, \( \Delta t = t_{ba} - t_n \). This approach is consistent with the derived radius factor expression and does not assume a constant Weber number. However, both of these approaches underpredict the radius factor (and thus overpredict \( N_{br} \)) in cases of droplets which have not been broken up for a long time. For example, consider a droplet which during the course of its traversal through the domain, does not distort much and hence \( y \sim 0 \). This may be if the Weber numbers it encounters along its trajectory are low (typically < 8). This droplet now encounters a flow region with a reasonable Weber number, say 20. Since \( K_{br} \) depends on the droplet properties and the Weber number, it has a reasonable value as well. However, \( t_{bu} \) in this case can be large, since the droplet has been unbroken throughout its lifetime. This leads to radius factor values which can be very low. Suppose that the radius factor value is 0.1; this implies that \( N_{br} = 1000 \). Thus, we have a droplet breaking explosively even though the Weber number is still fairly low. In case of both aforementioned approaches, there are multiple cases of erratic explosive breakups which are physically inconsistent. Additionally the expected model response in terms of predicting radius factor is shown in Figure 4.1. This clearly shows a one to one relationship between the radius factor and the Weber number which makes physical sense. Every droplet undergoing breakup at a given value of Weber number should produce the same radius factor, and thus split into the same number of product droplets. However, this becomes invalid if \( K_{br} \) is frozen or calculated based on numerical integration.
These problems lead to the third approach where we assume that $t_{bu}$ and $t_{bu,r}$ are two separately calculated quantities. The definition of $t_{bu}$ is exactly as mentioned in the previous section, calculated by the TAB model. However, $t_{bu,r}$ is calculated using Equation 4.6, where $y, \dot{y} = 0$ and $We$ is equal to the Weber number at breakup. Physically, this is the time required for the breakup of an imaginary droplet (with the same radius and material properties as the physical droplet undergoing the breakup process) with a constant Weber number throughout its lifetime. Note that for any droplet breaking up at the same Weber number, $t_{bu,r}$ will be equal.

This interpretation thus naturally leads to a one-to-one correspondence between the Weber number and the radius factor which is physically consistent and is in agreement with Figure 4.1. Additionally this also explains how Equation 4.9 reduces to 4.11, since $t_{bu,r}$ is calculated assuming a constant Weber number. A core concept of the TAB ideology is that droplet breakup depends on the history of the droplet. This is still preserved, since when the droplet breaks up is determined by $t_{bu}$ and thus the droplet history. However, the number of product droplets created is dependent solely on the Weber number at breakup.

### 4.2.2 Variation of $K_{br}$ with Weber Number

The radius factor definition used in Equation 4.11 forms the basis for the ETAB, CAB and Modified CAB (MCAB) models. These models primarily vary in their description of $K_{br}$ as a function of Weber number and the number of breakup regimes they account for.

In case of ETAB,

$$
K_{br} = \begin{cases} 
    k_1 \omega, & \text{if } We \leq We_t \\
    k_2 \omega \sqrt{We}, & \text{if } We > We_t
\end{cases}
$$

(4.13)
where both $k_1$ and $k_2$ are equal to $1/4.5$ and the transition Weber number $W_{e_t}$ is experimentally set to be 80. To maintain continuity between the regimes in terms of radius factors, the ETAB model uses a smoothening function to be multiplied to $k_1$,

$$\left\{ \frac{k_2}{k_1} \left( \sqrt{W_{e_t}} - 1 \right) \right\} \left( \frac{W_e}{W_{e_t}} \right)^4 + 1$$  \hspace{1cm} (4.14)

In the case of the CAB model [28], we have

$$K_{br} = \begin{cases} k_1 \omega, & \text{if } W_e \leq W_{e_{b,s}} \\ k_2 \omega \sqrt{W_e}, & \text{if } W_{e_{b,s}} < W_e \leq W_{e_{s,c}} \\ k_3 \omega W_e^{3/4}, & \text{if } W_e > W_{e_{s,c}} \end{cases}$$  \hspace{1cm} (4.15)

The definition of $K_{br}$ implemented in the droplet engine is as defined by the CAB model, given by Equation 4.15. For the CAB model [28], $W_{e_{b,s}}$ represents the regime dividing number from bag to stripping breakup and is equal to 80. $W_{e_{s,c}}$ represents the regime dividing number between stripping and catastrophic breakup and is set to 350. The paper states that the first constant $k_1$ is set to 0.05, while $k_2$ and $k_3$ are defined so that $K_{br}$ is continuous at the regime dividing Weber numbers. However, that does not clearly give the exact functional form of $k_1$, $k_2$ and $k_3$. As a result, a curve fit for $k_1$ was derived based on the radius factor results reported by the CAB paper [28] as shown in Figure 4.1.

$$k_1 = 0.05 + \frac{W_{e_{k1}} - 10}{W_{e_{b,s}} - 10} \left( 0.087 \sqrt{W_{e_{b,s}}} - 0.05 \right)$$  \hspace{1cm} (4.16)

In Equation 4.16, $W_{e_{k1}}$ is given as:

$$W_{e_{k1}} = \min(W_e, W_{e_{b,s}}) \quad ; \quad W_{e_{k1}} = \max(W_{e_{k1}}, 10)$$  \hspace{1cm} (4.17)

This curve fit was determined using the known values of $k_1$ at $W_e = 10$ and $W_e = W_{e_{b,s}}$, which were derived from the plot. In between these values, a functional form which varied $k_1$ to provide the best possible agreement with the radius factor plot of the CAB model, was postulated. Figure 4.2 shows the comparison between the radius factors of the actual CAB model, and the designed curve fit. The exact values as specified by the CAB model are shown by the scatter plot (in blue) while the functional variation of $k_1$ as given by Equation 4.16 is shown by the dashed line (in red).
4.2.3 Calculating the Normal Velocity components

The newly created product droplets take the positions and velocities of the parent droplets. However the TAB model postulates that the vibrational energy of the parent droplet results in the generation of a normal component which must be added on to the product droplet velocities. These normal components also ensure that each product droplet follows a different path. The magnitude of the normal component is given by:

\[ |\mathbf{v}_n| = 0.5 \ a \ \dot{y} \]  

(4.18)
Here, \( a \) denotes the radius of the parent droplet and \( \dot{y} \) is the value of the deformation rate at breakup. The directions of the velocity components are defined to conserve momentum. A direction vector is created for each of the product droplets in the plane perpendicular to the velocity of the original parent droplet. These vectors are so arranged that their net sum equals zero. An orthogonal basis is first created in the plane perpendicular to the parent droplet velocity vector \( v_p \). The first basis vector \( b_1 \) is determined by taking the cross product of \( v_p \) with a randomly generated vector and normalizing the result. The second basis vector \( b_2 \) is then created as:

\[
b_2 = \frac{b_1 \times v_p}{|b_1 \times v_p|} \tag{4.19}
\]

The direction vectors for each product droplet are then defined by rotating the basis vector \( b_1 \) by multiples of the angle \( \theta = \frac{2\pi}{n_d} \) where \( n_d \) denotes the total number of product droplets generated by the parent droplet under consideration. These unit direction vectors are then multiplied by \( |v_n| \) to get the normal components to be added to each droplet.

For the \( i^{th} \) normal velocity vector, corresponding to the \( i^{th} \) newly created droplet, we have

\[
v_{n,i} = |v_n| \left[ b_1 \cos \left( (i - 1)\theta \right) + b_2 \sin \left( (i - 1)\theta \right) \right] \tag{4.20}
\]

Note that \( i \) loops from 1 to \( n_d \), the number of newly generated droplets. Note also that this method assumes uniformly-sized child droplets.
Chapter 5

Droplet Vaporization

Droplet vaporization is an important phenomenon observed in spray applications. The effects of droplet vaporization become especially pronounced in high speed and high temperature flows typically seen in turbulent combustion. The droplet vaporization model used in the scope of this work is the Langmuir Knudsen-1 (LK-1) model as presented in [18] (also referred to as 'M7' in the cited reference). This model is based on the Langmuir-Knudsen law and accounts for non-equilibrium evaporation processes. However, this model ignores temperature variation inside the droplet. As a result, a single temperature is associated with any given droplet.

5.1 Governing Equations

The governing equations used in [18] to evolve the droplet temperature $T_d$ and mass flow rate due to droplet evaporation $\dot{m}_d$ are as follows:

$$\frac{dT_d}{dt} = \frac{f_2 Nu}{3 Pr_G} \left( \frac{\theta_1}{\tau_d} \right) (T_G - T_d) + \left( \frac{L_v}{C_L} \right) \frac{\dot{m}_d}{m_d} - H_{\Delta T}$$  \hspace{1cm} (5.1)

$$\frac{dm_d}{dt} = -\frac{Sh}{3 Sc_G} \left( \frac{m_d}{\tau_d} \right) H_M$$  \hspace{1cm} (5.2)

where, $\dot{m}_d$ is negative for evaporation, $T_G$ is the local carrier gas temperature, $L_v$ is the latent heat of vaporization, $\theta_1 = C_{p,G}/C_L$ is the ratio of the gas heat capacity (at constant pressure) to that of the liquid phase. Additionally $\tau_d = \rho_d D^2/(18 \mu_G)$ is the particle time constant for Stokes flow where $D$ is the droplet diameter. Furthermore $f_2$ represents the correction to heat transfer due to evaporation, $H_{\Delta T}$ is the correction to account for internal circulation and heat transfer effects inside the droplet (i.e. finite thermal conductivity) and $H_M$ represents the specific driving potential for mass transfer.
The above formulation uses Prandtl ($Pr_G$) and Schmidt ($Sc_G$) numbers defined in terms of the free stream gas values (as denoted by the subscript) with the following formulations:

\[
Pr_G = \frac{\mu_G C_{p,G}}{\lambda_G} \\
Sc_G = \frac{\mu_G}{\rho_G \Gamma_G}
\]  

(5.3)

For heat and mass transfer, the widely used Ranz and Marshall correlations for the Nusselt ($Nu$) and Sherwood ($Sh$) numbers are used [18].

\[
Nu = 2 + 0.552 Re_d^{1/2} Pr_G^{1/3}
\]  

(5.4)

\[
Sh = 2 + 0.552 Re_d^{1/2} Sc_G^{1/3}
\]

where $\mu_G$ is the dynamic viscosity, $\lambda_G$ is the thermal conductivity, $\rho_G$ is the density of the carrier gas phase respectively.

### 5.2 Implementation Details

Two definitions of Reynolds number are used in the description of the heat and mass transfer equations. The droplet Reynolds number ($Re_d$) is defined based on $v_r$, which is the magnitude of the slip velocity between the droplet velocity ($v_p$) and the continuous flow velocity interpolated to the droplet location ($v_I$). The blowing Reynolds number ($Re_b$) is defined using the blowing velocity $u_b$ which is defined in Equation 5.6.

\[
Re_d = \frac{\rho_g |v_r| d_p}{\mu_g} ; \quad v_r = v_I - v_p
\]  

(5.5)

\[
Re_b = \frac{\rho_g u_b d_p}{\mu_g} ; \quad u_b = \frac{-\dot{m}_d}{\pi \rho_g D^2}
\]  

(5.6)

Since the LK-1 model used in this work ignores heat transfer and temperature variation inside the droplet, $H_{\Delta T}$ is zero. The mass transfer potential $H_M$ is defined in the LK-1 model in terms of the non-equilibrium Spalding numbers ($B_{M,neq}$) for mass transfer.

\[
H_M = ln(1 + B_{M,neq})
\]  

(5.7)
where $Y_s$ represents the vapor mass fraction at the droplet surface while $X_s$ represents the mole fraction. Similarly $Y_G$ denotes the free stream mass fraction of vapor. $\theta_2 = W_C/W_V$ is the ratio of molecular weights of the carrier phase (denoted by subscript $C$) and the vaporizing liquid. Note that the Spalding mass transfer numbers, vapor mass fraction and vapor mole fractions are all calculated under non-equilibrium conditions as highlighted by the subscript $neq$.

\[ Y_{s, neq} = \frac{X_{s, neq}}{X_{s, neq} + (1 - X_{s, neq})\theta_2} \]  \hspace{1cm} (5.9)

\[ B_{M, neq} = \frac{Y_{s, neq} - Y_G}{1 - Y_{s, neq}} \]  \hspace{1cm} (5.8)

Here, $X_{s, eq}$ denotes the mole fraction of the vapor under equilibrium conditions, $P_{atm}$ is atmospheric pressure and $P_G$ is the pressure of the carrier gas at the droplet surface, $\bar{R}$ denotes the universal gas constant, $L_V$ is the latent heat of vaporization of the droplet, $T_B$ is the boiling point of the liquid and $T_d$ is the temperature of the droplet. $L_K$ is the Knudsen layer thickness and $\alpha_e$ is the molecular accommodation coefficient (assumed equal to unity). Finally, the non-dimensional evaporation parameter is represented by $\beta$ as follows:

\[ \beta = -\left(\frac{3Pr_G \tau_d}{2}\right) \frac{\dot{m}_d}{m_d} \frac{Pr_G Re_b}{2} \]  \hspace{1cm} (5.13)

Note that the surface mole fraction under non-equilibrium varies from the equilibrium value by the product of the Knudsen layer thickness and the evaporation parameter normalized by the droplet radius. As $\beta \to 0$ then $X_{s, eq} \to X_{s, neq}$.

Different models use multiple correlations to calculate the correction term $f_2$. The complexity and non-linearity involved depends on the range of evaporation rates used to corelate the data. In general there are no agreed upon correct models. However empirical curve fits are not quite necessary because the quasi-steady solution of the gas field equations coupled to the
droplet surface boundary conditions leads to an analytical solution [18], given below.

\[ f_2 = G = \frac{\beta}{e^\beta - 1} \]  

(5.14)

Equations 5.2 and 5.1 are integrated using a first order forward Euler explicit scheme. The equation for \( \dot{m}_d \) shows an implicit relationship due to the dependence of \( \beta \) on \( \dot{m}_d \). For the initial timestep, however, a few Picard iterations (< 5) may be used to help converge \( \dot{m}_d \) to the correct value. However, for the rest of the droplet lifetime, Miller et. al. [18] clarifies that \( \beta \) may be calculated using \( \dot{m}_d \) from the previous timestep. This is due to the logarithmic form of \( H_M \) used. Also, the non-equilibrium contribution due to the \( 2L_k\beta/D \) term used in Equation 5.11 is generally small for larger droplets. For smaller droplets, this contribution may be greater; however \( \beta \) is, in general, a slowly varying parameter so that even for smaller droplets, using \( \dot{m}_d \) from a previous timestep to evaluate \( \beta \) is adequate.

### 5.3 Selection of properties

The calculation of mass and heat transfers is dependent on the material properties used. These properties are functions of temperature and mass fractions of the vapor. Additionally selecting the exact temperature for evaluation is a challenge due to the difference in temperature of the carrier gas and the droplet. As a result, properties may be calculated at a reference temperature \( (T_R) \) and mass fraction \( (Y_R) \).

\[
\begin{align*}
T_R &= T_{d,s} + A(T_G - T_{d,s}) \\
Y_R &= Y_s + A(Y_G - Y_s)
\end{align*}
\]  

(5.15) (5.16)

Here the far field carrier phase temperature and vapor mass fraction are referenced by subscript \( G \), while surface properties at the droplet are referenced by the subscript \( s \). \( A \) is defined by the well known 1/3rd rule, thus \( A = 1/3 \) is used in Equation 5.15. Similarly expressions for calculating properties varying with diffusion are given in the paper by Miller et. al. [18]. However re-calculating these properties for each time step may be computationally expensive. As a result these properties are computed only once at the start of the simulation. The reference temperature may be defined to either be the estimated wet bulb temperature or the boiling temperature. The paper by Miller et. al. also shows that evaluation at the wet bulb temperature yields decent agreement with experimental results and hence is used in the presented work as well.
Due to the lack of accurate theoretical means of estimating the wet bulb temperature ($T_{WB}$), the following correlation is used to calculate it in deg K [18],

$$T_{WB} = 137 \left( \frac{T_B}{373.15} \right)^{0.68} \log_{10}(T_G) - 45$$  \hspace{1cm} (5.17)

Material properties for the carrier gas species are evaluated once at $T_{WB}$ at the start of the simulation and used. Liquid properties are assumed to remain constant. The expressions used for calculating temperature dependent properties are specified in [18].
Chapter 6

Immersed Boundaries

The CFD solver has the capacity to handle immersed boundaries (IBs). IB methods allow for simulation of very unique engineering problems. IB techniques can also be used to simplify the rendering of complicated geometries. Examples presented in [21] and discussed later include the rendering of an aircraft interior and the rendering of a collapsed building. References [4, 5] present several examples of the use of NCSU’s IB techniques to model complicated flows induced by human activity and motion events. As an example, the combination of the CFD solver and droplet tracking engine were used to simulate contaminant transport in an aircraft, with patients and other travelers. The people and the internal components in the aircraft were modeled using the immersed boundary method. The flow inside the aircraft was driven by the HVAC system and thermal plumes of the people. The sneezing events of various patients were simulated as a droplet cloud which was then tracked by the droplet tracking engine to understand droplet concentrations and depositions on various objects, rendered as IBs. Methodologies developed to capture these events including droplet deposition on IBs and droplet injections off IBs are discussed in detail in this chapter.

6.1 Data Structures for Immersed Boundaries

IBs are rendered as stereolithography (STL) files as shown in Figure 6.1. These contain information about the rendered surface as a set of points, edges and triangulated faces as seen from the figure. Note that IBs may be stationary or may move through the simulation. In cases of motion, there are multiple copies of the IB object representing the object’s position at various timesteps, which are sequentially imported through time. Additionally, there may be multiple IBs in the domain. The entire information regarding an object can be captured by means of an STL description.

The CFD code resolves IBs using different data structures which carry the object information
in terms of vertices, edges and faces so that they may be queried as required. It also uses field descriptions of each IB along with some auxiliary information which returns useful data for processing IBs. The basic description of any IB is via the generation of two field quantities: a signed distance function and a Heaviside function. The signed distance function at any point in the domain gives the distance of that point from the nearest IB. The signed distance function is positive outside the IB, zero at the IB surface and negative inside. The Heaviside function is 1 inside the IB and 0 otherwise. Thus, any IB in the domain divides the mesh into three categories of cells:

1. *interior cells*, which are inside the IB
2. *banded cells*, which are outside the IB but have at least one neighbor in the interior
3. *field cells*, which are completely outside the IB

The signed distance and Heaviside functions are calculated by the solver every time step using an approximate nearest neighbor search algorithm. This step is then followed by an element-wise search to determine nearest vertices, edges and faces. The Navier-Stokes equations are solved in the field cells, and interpolation methods derived from classical boundary layer theory are used in the band cells to connect the solution in the field cells to the motion of the immersed object. Further details may be found in [4] and [5]. The droplet tracking engine was modified to be able to recognize IBs as well as meshed surfaces as locations for droplet injection and deposition.

The IBs in the domain are described using arrays of specially defined data structures. The vertices, edges and faces forming each IB are stored as lists of defined types *vertex*, *edges* and *elem* respectively. Another defined data-type *surfpar* keeps track of the overall information of an IB. There are other auxiliary data types which keep track of pointers to specific information of various IBs as well. Additionally there are field arrays *lpri* and *nlist* which store information regarding the nearest IB to any given cell in the domain. Based on these specialized data structures, there are multiple intrinsic functions which report specific information necessary to use IBs for both CFD and droplet tracking. For example, *surf_info* can return the information about the nearest object element (vertex, edge or face) given the information of a cell ID and the nearest IB. Some of these intrinsic functions have been modified to enable coupling with the droplet tracking engine.

### 6.2 Droplet Interaction with Immersed Boundaries

Droplets interacting with immersed boundaries are currently made to deposit on said IB. To this end, the information of the IB and the IB vertex nearest to the droplet is linked, thereby
simulating deposition. It is important to note that this IB may move during the simulation and thus the deposited droplet must be moved along with it. This is achieved by the droplet tracking engine by tagging droplets deposited on IBs separately using the state flag (See Chapter 8 for details about the state flag). While these droplets are not tracked like the other advecting droplets, their positions are updated each timestep using the linked IB and vertex information.

During the course of droplet tracking, the information regarding the cell which contains a given droplet is easily accessible. Using these cell IDs, the code determines the values of the signed distance function and the Heaviside function discussed in the previous section. These values are checked for every droplet at each timestep. If the Heaviside function value is 1 and the signed distance function is less than 0, it implies that the droplet is just inside an immersed boundary, and the droplet interaction with an IB is triggered.

Based on the cell IDs of the cell containing the droplet, the nearest IB is located. Using this information the data regarding the nearest object element can be determined with the help of surf_info. Using some of the data structures discussed in the previous section, the index of the vertex, edge, or face closest to the droplet is determined. This information is then stored as a part of corresponding droplet entry in the the cellids array. This links the information of the nearest vertex of the IB that the droplet has interacted with to the droplet itself. The deposited droplet is assigned a unique state flag value by defining cellids(6, np) to 5, where np is the droplet index on that processor. Note that this droplet remains on the same processor throughout the simulation. Its position will be updated each iteration in case the linked IB moves.
6.3 Droplet Injection from Immersed Boundaries

The CFD code reads in and demarcates certain points for injection at the time of initialization. In case of the specific simulation for which results are presented in Chapter 9, injection zones were the mouths of people which would sneeze during the simulation. This was done by creating a sphere near the patient’s mouth. The intersection points with the person’s IB object and the sphere allowed the code to tag object elements for injection. Primarily, the injection process creates new entries of droplets which requires information regarding droplet position, velocity, diameter and material properties. The position for droplet injection is defined by finding the specific cells containing the tagged vertices. This is also done using the aforementioned surf_info. The number of droplets to be injected is an input parameter. Based on this and the number of tagged cells, new droplet entries are created. The same surf_info subroutine also provides information about surface normals in the injection region. Using an input velocity magnitude and the surface normals, the velocities at the injected droplets may be prescribed.

Since the specified simulation required multiple patients sneezing, an additional input file provided data about the timing of sneezing events associated with each person. This input information was read in and suitably used to trigger droplet injection corresponding to the correct IB at the right time. To successfully analyze the droplet distribution and deposition process, it is necessary to know the source of the injecting IB. The presence of multiple injecting IBs makes this process challenging. For this reason each droplet has an associated IB source flag stored as a part of the cellids array. This allowed the determination of the source of every injected droplet at any point during the simulation. The code also provides for special injection processes where the source may be split across multiple processors. In such a case, the total number of cells used for IB injection are first determined using mpi_allreduce which is an intrinsic function in OpenMPI. The total number of injecting droplets is then divided among all these cells to trigger injection.
Chapter 7

Parallelization

Simulation of engineering problems requires successfully tracking a large number of droplets, usually of the order of millions. This is made practically possible by using parallelization. During the course of the simulation droplets are advected and thus flow out of the blocks they were previously in. As discussed in preceding chapters, droplet search and eventual interpolation and integration requires the knowledge of the local block identification number (ID) in which the droplet is present. In a multi-block environment, this means that having a complete knowledge of the connectivity of all blocks is necessary. Additionally in a multi-processor environment these blocks may be mapped to different processors. This information must also be known so that relevant droplet information may be packaged and sent to the right processor. This chapter discusses in detail the pre-processing required to establish a grid connectivity structure for the droplet tracking engine as well as implementation details of the parallel algorithm.

7.1 Basic Objectives and Challenges

The droplet tracking engine is parallelized using OpenMPI, working on a distributed memory architecture. Working with a distributed memory architecture gives a higher amount of flexibility and also makes the droplet tracking engine scalable. For larger grids, or for injecting higher number of droplets, simply increasing the number of processors can usually provide reasonable performance. Note that the scalability of the engine has not been tested and thus in this scope, the term 'scalable' does not imply its formal definition as used in assessing parallel algorithms. A downside of distributed memory architectures as the name suggests, is the fact that storage space is distributed across processors. As a result of this, each processor can see only a certain part of the domain which necessitates communication across processors. Similarly, such an architecture makes it difficult to balance computational loads since it requires an independent optimization process balancing cost shared by each processor and the cost for communication.
Additionally this paradigm makes the process of designing the algorithm challenging and may impart certain restrictions to the maximum speedup possible. In iterative algorithms all processors usually must synchronize at the end of each iteration. Thus the time required for any given iteration will be determined by the slowest processor. However, despite these drawbacks, this architecture provides excellent performance overall if used correctly, especially for CFD applications.

In a general multi-block grid, for a given block we have a total of 26 neighbor zones. The term neighbor zone of a block can be defined as the regions adjacent to the block, sharing at least one vertex with it. Based on the number of shared vertices these zones are named as follows:

- **Corner** neighbor zone: 1 common vertex
- **Edge** neighbor zone: 2 common vertices
- **Face** neighbor zone: 4 common vertices

These zones have been illustrated in detail in Figure 7.1.

![Neighbor zones for a block](image)

**Figure 7.1**: Neighbor zones for a block

Note that each neighbor zone may have one or more blocks. In cases of domain boundaries,
there may be no blocks in a neighbor zone. However this idea is instrumental in setting up the grid connectivity used by the droplet tracking engine. As seen from Figure 7.1, the 26 neighbor zones are split as follows:

- 6 face neighbor zones (shown in red)
- 12 edge neighbor zones (shown in blue)
- 8 corner neighbor zones (shown in green)

A droplet in a given block may be advected into any one of the neighbor zones. Note that some of these neighbor zones may be empty due to the presence of boundaries such as walls, outlets or other boundary conditions such as symmetry. For the purpose of this work, these block interfaces are termed as boundaries while block interfaces which connect two blocks are referred to as interfaces. The grid connectivity information must therefore capture the relative positions of blocks in all these zones along with boundary conditions. Furthermore, neighboring blocks may be mapped to different processors, and this information must be recorded as a part of grid connectivity. There are certain differences in the connectivity information required by the CFD solver and the droplet tracking engine. For example, the CFD solver requires the orientation of the face neighbors which is not necessary for the droplet engine. On the other hand, the droplet tracking engine requires the knowledge of blocks in the edge and corner neighbor zones along with face neighbors. As a result, a separate grid connectivity file is generated for the droplet tracking engine. Also note that blocks have two identification numbers: a global ID which is unique to each block and specified as a part of the input file to the code, and a local block ID which is unique to the processor the block is mapped to.

The droplet tracking engine uses the same domain splitting as the CFD solver. There are multiple approaches to handling parallelization of droplet engines like using master-slave configurations or having separate processors only handling the droplet tracking process. However, to facilitate transient droplet tracking where the CFD code and the droplet engine interact every time step, using the same domain splitting for both works best. For each iteration droplets must know the surrounding flow field to calculate the forces acting on them. The flow field data for any part of the domain exists only on a single processor - the processor that block is mapped to. As a result, any droplets in that block are naturally assigned to that processor. This approach can unfortunately create load imbalance. However, these effects have been neglected in the scope of the work presented in this thesis.

The nature of having an Eulerian flow description means that physical domains split in space do not need to change over time. This is because the computational cost for a certain domain mapped to a specified processor can be pre-computed and invariant. As a result, the Lagrangian definition of droplet tracking offers certain unique challenges:
1. Need for information of all neighbor zones of any block, not just face neighbors

2. Transient droplet loading in any part of the domain

3. Lack of preset communication patterns

4. Possibility of multiple blocks in a single edge-corner neighbor zone

Multiple blocks in a neighbor zone are a reasonably frequent occurrence for complex geometries. Allowing this greatly enhances the flexibility of multi-block grids and is therefore essential. The maximum number of allowable blocks in a given zone is an input parameter used by the droplet tracking engine. Note that the presence of multiple blocks in a given zone significantly complicates both the generation of grid connectivity and parallel implementation. Both of these will be specifically addressed in the following sections.

### 7.2 Grid Connectivity for Droplet Tracking

Generating grid connectivity is a part of the pre-processing step. The code uses three inputs for this purpose:

1. Solver input file
2. Grid files
3. The connectivity file generated used by the CFD solver, `gridpro.conn`

As shown in Figure 7.1, there is a very specific numbering used to identify each neighbor zone, with respect to the \((i,j,k)\) orientation of the block (marked in black). During the droplet tracking process, a droplet which crosses the block interface will be found in one of the ghost cells. The cell identification number of this cell, along with the pre-generated grid connectivity should provide information necessary to determine what block or blocks the droplet has now moved into or possibly what boundary conditions it encounters. The specific information on handling boundary conditions is provided in Chapter 8. In case of droplets crossing ‘interfaces’ the connectivity must also provide information about the processor to which this information must be sent.

A three-stage linked list is established to maintain and correctly reference connectivity information as illustrated in Figure 7.2. At the first stage, based on the ghost cell ID of the droplet the number of the zone (as shown in Figure 7.1) is first determined. Rather than using multiple sets of \(if\) conditions, this has been done by mapping the cell position relative to the block based on its indices to a \(3 \times 3 \times 3\) matrix which links the cell to the number of the zone.
directly. Thus, we now know the neighbor zone ID \( m \) and the global block ID \( n \). Based on this information, we can reference the second stage of the linked list as follows:

\[
i = 26 \times (n - 1) + m
\]  

(7.1)

In Equation 7.1, the number 26 comes due to the fact that each block has 26 neighbor zones. The second stage of the linked list is a one dimensional array, whose index \( i \) will provide the pointer to the third and final stage of the list. The pointer will give the index of the start of the list of blocks in that neighbor zone \( m \) for the given block \( n \) that is listed in a file which forms the third stage of the linked list. Note that this second stage allows us to handle the presence of multiple blocks in a given neighbor zone. The third stage provides the information about the boundary condition, the ID of the processor in case of an 'interface' and finally the local block ID. Additionally it also provides data about a \textit{shift} parameter which is useful in the case of periodic boundary conditions.

\[\text{Figure 7.2: 3 stage linked list for grid connectivity}\]
Boundary conditions are encoded by means of integral values as follows:

- Interface/Periodic: 2
- Wall: -1
- Symmetry: -2
- Outlet: 0

The solver input file has information about the number of blocks, their dimensions along the \((i, j, k)\) axes, global block IDs, and the identification number of the processor they are mapped to. Using this information, the actual grid data which includes the \((x, y, z)\) co-ordinates of each point in every block, is read. These two pieces of data allow the code to delineate the eight corner vertices of every block. The code then uses nested loops over each block and compares these end vertices to find neighbors in each zone. Based on the number of common vertices, the list neighbor blocks for each block is assembled.

- For face neighbor blocks: 4 common vertices
- For edge neighbor blocks: 2 common vertices
- For corner neighbor blocks: 1 common vertices

This is naturally concurrent with the definition of the corresponding face, edge and corner neighbor zones which contain these blocks. Also note that the orientation of the neighbor blocks with respect to the core block is used while determining the ID of the neighbor zone in which the block resides. To account for multiple blocks in a zone, the code uses a buffer which stores the list of all blocks in that neighbor zone. After the 'interfaces' are thus resolved, the grid connectivity code will then look at the \textit{gridpro.conn} file to assign the necessary boundary conditions. By default, any block interface that is not adjacent to any other block is termed to be a wall. This is then re-written based on the input of \textit{gridpro.conn}. The code will never re-write interfaces, instead raising error flags since that essentially points to a mistake. Similarly, boundary conditions corresponding to face neighbors are not allowed to be re-written either, followed by an error flag. It is important to note that this boundary condition information is pertinent to the face neighbors alone. However, for droplet tracking, these boundary conditions need to be extended to corner and edge neighbors. The boundary condition tags used for each boundary are thus not random; they are issued in descending order of priority. For example, consider the case of an edge with one adjacent face set to be a wall (BC type 1) while the other face is set as an interface (BC type 2). The boundary condition edge neighbor zone is set to be a wall since \(1 < 2\). Similarly in case of corner neighbor zones, there may be three separate
possibilities for boundary conditions due to three separate flags on the three faces forming said corner. The boundary condition assigned to the corner neighbor zone will then be the one with the lowest integer value or the highest priority. Finally, all of this information is printed out into two files which form the second and third stage of the linked list. This information can then be read in during the course of the simulation. The first stage of the linked list, which is a $3 \times 3$ mapping function described earlier in this section, is part of the code itself.

### 7.3 Parallel Implementation

During every iteration, droplets which cross interfaces are tagged and the information about the new block and the new processor its information should now be mapped to is noted. Similarly, there are multiple auxiliary arrays which are used to keep track of the droplet exchange process. Finally, all this information is bundled and exchanged. Based on this exchanged information each processor will reallocate memory based on droplets sent to other processors as well as received from them. The exact process and data structures used to achieve this is discussed in the section in detail.

For every iteration, the code loops over each of the droplets for search, advection and integration. When such an advected droplet is found in a ghost cell, the necessary boundary condition associated with it is determined. This information is maintained by means of two arrays: $cellids$ and $cidsenddata$. Both these arrays are $n \times npar$ where $npar$ is the total number of particles being tracked by a given processor. $n$ for $cellids$ is 8 and for $cidsenddata$ is $2 \times nbfr + 1$. The $nbfr$ is an input parameter which gives the maximum number of blocks in a zone. The $cellids$ array has a state flag which monitors which droplets must be transferred at the end of the current iteration. The $cidsenddata$ keeps an account of the number of blocks and processors this information is to be sent to. It also makes note of the processor ID to which the droplet data must be sent and the corresponding local block ID on the new processor. This information is pulled from the grid connectivity linked list explained in the previous section. Simultaneously another array called $sendlist$ keeps track of the total number of droplets to be sent to each processor. Once the loop over all droplets is concluded, the $cidsenddata$ entry for each droplet is populated with the necessary exchange information. Based on the maximum number of droplets being sent ($binsize$) which can be found from $sendlist$, a new array is allocated, called $sendbin$. This array is a $binsize \times nproc$ sized array which makes note of the IDs of the droplets being sent to each processor. The $cidsenddata$ array is then used to compute $sendbin$. These three arrays are finally used by the transfer subroutine to package and send the droplet information.

For each droplet 15 pieces of information must be exchanged, as follows:

- Droplet position (3)
- Droplet distortion, distortion rate and droplet time used for droplet breakup modelling (3)

- Droplet velocities, temperature, vaporization mass flow rate and diameter (6)

- Droplet source, droplet break-up state, new local block ID (3)

The exact implementation of the droplet information exchange can be best explained by directly referring to the subroutine performing this operation, as shown below. Comments have been added to explain the significance of each part of the code.

```fortran
subroutine transferdata_test_a2av_intra(t,nprocs,m,pprop,ipos,cellids,sendbin,&
sendlist,cidsenddata,recvbuffer,point,it)

include "param.v2.0.inc"
include "common.v2.0.inc"
include "mpif.h"

integer, intent(in) :: t,nprocs, m,it
integer, intent(out):: point
integer,dimension(:,,:),intent(in) :: sendbin,cidsenddata,cellids
integer,dimension(:,,:),intent(in) :: sendlist
real*8, dimension(:,,:),intent(in) :: pprop,ipos
real*8,allocateable,dimension(:,),intent(out) :: recvbuffer

real*8,save,dimension(:,),allocateable :: val

integer :: npcount,proc_id,c1,c2,np,ierr

integer, dimension(nprocs) :: send_displ, send_count, recv_displ, recv_count
integer :: i,recv_size,send_size
logical :: count_error
integer :: ncheck

!Note: proc_id is the process number of the proc to which data is to be SENT. rank =
! proc_id -1.

! set the send_count/recv_count
send_count(1:nprocs) = sendlist(m,1:nprocs)
count_error=.false.
do i=1,nprocs
  if(send_count(i)<0) then
    write(*,*)"send_count<0", m, i,send_count(i)
    count_error=.true.
  endif
enddo
if(count_error) then
  stop"send_count_error"
endif

call MPI_ALLTOALL(send_count,1,MPI_INTEGER, &
  recv_count,1,MPI_INTEGER, MPI_COMM_WORLD, ierr)

send_displ(1) = 0
```

50
recv_displ(1) = 0

do i=2,nprocs
    send_displ(i) = send_displ(i-1)+send_count(i-1)
    recv_displ(i) = recv_displ(i-1)+recv_count(i-1)
enddo

send_size = send_displ(nprocs)+send_count(nprocs)
recv_size = recv_displ(nprocs)+recv_count(nprocs)

if(allocated(recvbuffer)) deallocate(recvbuffer)

! allocate memory here
allocate(recvbuffer(recv_size*indx_x))
allocate(val(send_size*indx_x))

! pack the data altogether
point = 1
do proc_id=1,nprocs
    if(send_count(proc_id)>0) then
        !if(proc_id.ne.m .and. sendlist(m,proc_id).ne.0) then
        npcoun = 1
        do c1=1,sendlist(m,proc_id)
            np = sendbin(c1,proc_id)
            if(c1>1) then
                if(c1>1 .and. np==sendbin(c1-1,proc_id)) then
                    npcoun=npcoun+1
                else
                    npcoun=1
                end if
            end if
            !The above has been done to account for the fact that the same particle
            !might have to be sent multiple times to the same processor.
            !This will happen if the multiple edge/corner blocks lie on the same
            !processor.
        ncheck = 0
        do c2=1,cidsenddata(1,np)
            if (cidsenddata((2*c2),np)== proc_id) then
                ncheck = ncheck+1
                val(point) = cidsenddata(2*c2+1,np)!Local block id
                if(ncheck==npcoun) exit
            end if
        end do
        val(point+1:point+6) = ipos(1:6,np)
        val(point+7:point+15) = pprop(1:9,np)
        val(point+16:point+17) = cellids(7:8,np)
        point = point+indx_x
        end do
    end if
end do

call MPI_ALLTOALLV(val,send_count*indx_x,send_displ*indx_x,MPI_REAL8,&
recvbuffer, recv_count*indx_x, recv_displ*indx_x, MPI_REAL8, MPI_COMM_WORLD, ierr)

deallocate(val)
point = recv_size*indx_x + 1 ! recover the same 'point' value as in other version.
return

end subroutine transferdata_test_a2av_intra

An important thing to note is that to handle multiple blocks in the same neighbor zone, duplicate copies of the same droplet are created and sent to the processors of both blocks. It is very difficult and possibly computationally costly to determine that which exact block the droplet enters after crossing an edge or a corner \textit{a priori}. While multiple entries can result in additional cost of search, the speed of the current droplet search algorithm is leveraged to solve the problem by elimination. In case the duplicate entry is not found in a block, that processor will simply delete said droplet and continue. Similarly, neighbor blocks may be on the same processor as the block sending the droplet information. In such cases a duplicate entry will be created for the same droplet, but with a new local ID. However this results in no extra computations as the old droplet entry is automatically deleted.
Chapter 8

Auxiliary Functions

The true complexity of the droplet engine when simulating large problems split across multiple processors comes from the challenge of book keeping and managing large amounts of data. The robustness and efficiency of the engine also depends significantly on handling these challenges effectively. Successful operation of the code is critically dependent on the code structure and general algorithm used. Similarly there are a variety of additional functions like handling droplet boundary conditions and droplet injection which are required for the overall simulation process. This section delves into some key aspects of the code structure and auxiliary functions.

8.1 General Algorithm and Data Structures

Droplet data on each processor is maintained using three 2-D arrays: \textit{ipos}, \textit{pprop} and \textit{cellids}. The first two arrays record the real valued droplet properties such as droplet position, distortion, velocity data, diameter etc. The third array keeps track of the cell which currently contains the droplet, the processor and local block ID, the droplet origin and the droplet state. During the course of the simulation, these arrays are reallocated based on the number of droplets being currently tracked by any given processor. Note that these arrays work in tandem with the \textit{cidsenddata} array mention in Chapter 7. The pertinent information regarding any droplet can thus be referenced by \textit{array(index, np)} (here, \textit{array} = \textit{ipos}, \textit{pprop}, \textit{cellids}) where \textit{np} is the droplet ID and \textit{index} maps to a specific property value.

8.1.1 Algorithm

An overview of the droplet tracking algorithm executed on each processor is as follows:
Algorithm 3 Overview of Droplet Tracking

Read inputs and initialize variables

while $t < t_{\text{max}}$ do

Solve CFD

\textit{mpi\_barrier} \{to synchronize processes\}

Calculate nodal flow values and vorticity

Inject droplets

for $it = 1$ to number of sub-its do

for all $i$ such that $1 \leq i \leq npar$ do

Droplet search

Interpolate field values to droplet location

Check for break up

Integrate

Droplet vaporization

if droplet close to block boundary then

if droplet advected into ghost cell then

Apply relevant boundary conditions

end if

end if

end for

Reallocate data for new droplets created during breakup

Package droplet data for exchange

\textit{mpi\_barrier}

Send and receive droplet data

\textit{mpi\_barrier}

Reallocate data for sent, received and deleted droplets

end for

end while

8.1.2 The state flag

As mentioned in the previous section, the \textit{cellids} array maintains a droplet \textit{state} flag. This is a crucial part of controlling the decision processes with respect to any droplet. The state flag determines if a droplet is relevant or to be deleted, records the boundary conditions encountered by the droplet, and performs some unique functions for newly injected droplets. By default a \textit{regular} droplet being tracked through the domain will have a state value of 1. In case this droplet is to be deleted due to wall intersections, outflow or if it is a duplicate copy which is
searched and not found in its assigned local block, it is assigned the state 0. In case of certain injection functions, newly injected droplets may need to be assigned velocities interpolated at their origins. This is particularly useful in cases where a droplet population is created on an injecting surface or a block volume, given a certain mass flux. In such cases the new droplets are identified by means of a special state. Once their velocities are determined using the same sequence of search and interpolation, they are re-tagged as standard droplets. This allows the incorporation of unique injection scenarios without making any change to the overall algorithm.

A similar concept is used in the case of droplets injected from immersed boundaries. In the first iteration, these droplets may be close enough to the ejecting surface that they get deposited back onto it. To prevent this, these droplets are tagged with special state flags but handled slightly differently. Another example is the ability of the code to record wall hits and keep track of droplets deposited on immersed boundaries. While these droplets are maintained as a part of the list on a processor, they are not actually tracked.

8.2 Handling Boundary Conditions

Similar to the CFD solver, droplets encounter boundaries as well, and these boundary conditions must be appropriately applied to the droplets. In Chapter 7, the interaction of droplets with block interfaces was addressed. In this section the interaction of droplets specifically with block boundaries is discussed in detail. The droplet engine recognizes the following boundary conditions:

1. Wall
2. Symmetry
3. Periodic
4. Outlet

As noted in Chapter 2 for all droplets close to the block boundaries, a droplet search is performed again, based on the new position of the droplets. If the newly determined cell is a ghost cell, then the relevant boundary condition is referenced from the grid connectivity. During this process, the information regarding the old cell which contains the droplet is stored before the new cell is determined. This old stored cell information is necessary for application of the wall boundary condition. Similarly, after a droplet is advected, its velocity is updated. However, its position is not updated. Instead, a separate vector stores the displacement for that droplet. This is done for the benefit of applying boundary conditions. In cases of interfaces, this displacement is simply added to the droplet, and its information is sent to the right processor.
However in case of wall and symmetry boundary conditions the entire displacement vector is required to find the exact point of intersection between the surface and the droplet path for that time step.

8.2.1 Wall

The droplet engine identifies walls as solid surfaces and stops the droplets at the wall. Currently there is no wall model that allows for reflection of a droplet off the wall. However the wall interaction is so computed that this functionality can be added easily. The code can save droplets deposited on the wall if that information is necessary based on the \textit{iwallsave} input parameter. Otherwise these droplets are simply deleted from the list of droplets on the corresponding processor.

Since the previous and the new droplet locations are known, the segment joining them must intersect some part of the domain boundary. To determine this exact location, the stored displacement vector is necessary. Additionally the knowledge of the old and new cells containing the droplet give the bounds on the ‘real’ boundary cells which will contain the face where the intersection occurs. This looping strategy is required because faces of the boundary may not be planar in a curvilinear grid. Based on the end points of each face, a planar equation is determined. A parametric formulation of the displacement vector is then used to determine the intersection point. The advantage of this approach is that it can report accurate intersection data and allows the calculation of incident angles with respect to the wall surface.

8.2.2 Symmetry

The application of the symmetry boundary condition is similar to that of the walls. Symmetry faces are assumed to be planar, which makes the process simpler as a loop over multiple faces is not necessary. Once the point of intersection with the planar surface is determined, based on the face normal, it is possible to reflect the droplet off the surface. In case of walls, only the droplet location needs to be calculated. However, since droplets encountering symmetry boundary conditions are essentially reflected back into the domain, their velocity vectors are reflected as well.

8.2.3 Periodic

During the grid connectivity generation process for droplets as described in Chapter 7 the code does not automatically determine periodic boundary faces. This is because geometrically periodic faces are not connected and hence have no common points. However, when the grid connectivity code reads in \textit{gridpro.conn}, periodic faces are recognized since they are additional faces marked as interfaces. Periodic faces are also assumed to be planar, like symmetry faces.
In the case of the CFD code, these partner blocks simply update their ghost cell information like a regular interface. For droplets, the application of periodic boundaries involves displacing the droplet physically from one face to the other. By looking at the co-ordinates of partner faces, the grid connectivity code determines a shift parameter by considering the difference in the end points of the marked faces. The shift parameter records the $x, y, z$ components of the displacement which must be applied to any droplet encountering that periodic boundary. This shift parameter is printed out as a part of the $pgridconn.dat$ file, which is the third stage of the droplet grid connectivity linked list.

Periodic boundaries are essentially treated like interfaces using the same boundary tag. When droplets cross a periodic interface the droplet tracking engine pulls up the local block ID and the processor corresponding to the new block, much like a regular block interface. Additionally the droplet is displaced by adding the shift to the droplet. It is then processed for exchange between processors as described in the previous chapter.

### 8.2.4 Outlet

Droplet flowing out of the domain through the outlet are simply allowed to leave the domain. They are tagged by setting the state flag parameter described above to 0. During the reallocation process after the exchange of droplet information, their records are simply deleted from the processor they that they were located on.

### 8.3 Droplet Injection

A variety of droplet injection functions are necessary for different simulations. In some cases droplets are ejected off a face based on a certain distribution while in some cases they may be injected to reflect a mass flux reported by the underlying CFD solver. Two different types of injection subroutines developed are discussed in this section. Note that droplet injection off immersed boundaries has been discussed in Chapter 6. In terms of implementation, a droplet injection process performs the following functions:

1. Copy existing droplet data
2. Reallocate droplet arrays to make room for new droplets
3. Create new droplet entries for position, velocity and other droplet properties based on input data and the injection type

A variety of parameters relating to injection such as injection type, number of droplets to be injected, the droplet diameter range and distribution, injection frequency and droplet velocities are provided through the input file.
8.3.1 Face Based Injection

Face based injection uses a separate input file which gives the global block numbers of the blocks whose faces are used for injection. Along with this, the face IDs of the injecting surfaces are also provided. Based on this input, the relevant processors tag the injecting faces in their domain and trigger the injection process. The input file also provides the total number of injecting faces and number of droplets to be injected each iteration. This information is used to determine the number of droplets to be injected off each face. The locations for injection can be randomized in each iteration to give better statistical results.

In the case of curvilinear grids, inlet faces may be curved. As a result, generating a random coordinate on or close to the face is challenging. Based on the face ID, the adjoining range of cells for this face is determined. The code then randomly chooses one of these cells in a loop over the total number of droplets to be injected off the face. Additionally, each cell is first mapped to a unit cube using the same isoparametric formulation used in trilinear interpolation. Inside the unit cube, random values for the \((\epsilon, \eta, \zeta)\) co-ordinate system are generated and mapped back to the cell as \((x, y, z)\) coordinates for the newly injected droplets. These droplets are initialized with other properties based on the input parameters. Note that the droplet search process for newly injected droplets is simplified for the first iteration. This is so because the IDs of the cells containing these droplets are already known.

8.3.2 Block Based Injection

Similar to face based injection, block based injection also uses a separate input file with a similar format as face based injection. The input file gives the total number of injecting blocks and their global block IDs. This automatically triggers injection of droplets on the relevant processors. In case of block based injection, an additional input comes from the underlying CFD solver as the total mass to be injected. This mass is contained in a thin disc-shaped cross-section of the injector, calculated based on the flow rate and the time step.

In one case where this procedure is applied, the experimental results indicate that the entering two-phase flow is core-annular in structure. The thickness of the annulus is found to be \(\sim 50\mu m\) and the size of the droplets in the core is assumed to be 10 to 20 \(\mu m\). Based on these values, the injection function calculates the number of annular droplets which may be fit to represent the annulus as a collection of droplets. The remaining mass is then randomly distributed in the interior of the disc as a collection of the smaller 20 \(\mu m\) (or 10 \(\mu m\)) droplets. The co-ordinates of all droplets are randomly generated to lie in the specified disc. In this case, since the injecting mass is in some sense replaced with the droplet field, the velocities assigned to the droplets must match the local velocities in the inflow. As a result, the velocities of the droplets are not set inside the injection function but later in the code as a part of the overall
droplet tracking algorithm. Note that these newly injected droplet are thus tagged separately using the previously discussed state parameter.

It is also possible to assume a droplet size distribution for the entering droplets and to sample this distribution to create the injected droplet population. The population size is determined by summing over the volumes of the sampled droplets until the target mass (acquired from the CFD code) is obtained. These droplets are then randomly distributed over the inflow exit plane and their velocities are determined by the interpolation process mentioned before. In Chapter 9, in case of droplet injection into subsonic cross-flow, a log normal initialization is facilitated by first sampling a log-normal size distribution with geometric mean equal to 7.5 $\mu m$ and a standard deviation in the logarithmic coordinate of 0.5. After each sampling event, the diameter is stored and the droplet volumes summed until the target volume is reached. This droplet population is then randomly distributed over the entire entrance plane. Similarly, droplets may be sampled from monodisperse, uniform and Gaussian distributions, based on a user-selected input parameter.
Chapter 9

Results

In this chapter, results from the application of the droplet engine are presented. Earlier validation studies emphasizing droplet responses to gravity and drag were presented in [7]. The current studies focus on the practical applications of the engine for several problems of interest, but also cover validation studies relating to droplet breakup and vaporization.

Results are presented for the following cases:

1. Aerated liquid injection into quiescent environment
2. Contaminant transport in a C-17 aircraft
3. Air assisted liquid atomization (droplet breakup validation)
4. Planar aerated liquid jet injection with droplet breakup
5. Injection into subsonic cross-flow
6. Single droplet vaporization (droplet vaporization validation)

Cases 1 simulates aerated liquid injection into a quiescent atmosphere in detail, primarily focusing on droplet response to high pressure injection and the surrounding supersonic flow. These results are then qualitatively compared to experimental data. Case 2 models contaminant transport in a confined space, highlighting the coupling of the droplet engine with immersed boundary methods. Cases 3 - 5 showcase results with droplet breakup. The droplet breakup model used in the engine is validated by means of experimental data for air assisted atomization of a single droplet in Case 3. Case 4 is a simplified version of Case 1: a planar aerated liquid injection with droplet breakup to provide an evaluation of the response of the breakup model in high speed flow. Droplet injection in a cross flow is investigated in Case 5; results are presented with and without breakup and compared to experimental data. Finally, Case 6 provides an initial validation of the droplet vaporization model, focusing on single droplet evaporation.
9.1 Aerated Liquid Injection into Quiescent Environment

Aerated-liquid atomization is the process of injecting small amounts of aerating gas into a primary liquid stream to accelerate primary atomization [12–15]. In simulating aerated-liquid injection processes, REACTMB initially tracks the evolution of large bubble-like structures, using a volume of fluid method [1]. These structures emerge as a result of injecting a small amount of vapor (a gas-to-liquid mass ratio of $\sim 4\%$) into a co-flowing liquid stream. The two-phase mixture then accelerates rapidly as it moves through a small orifice. The displacement effect of the aerating gas pushes much of the liquid toward the walls of the nozzle, resulting in a core-annular structure. It is assumed that the dispersed-phase flow within the core consists of small droplets which are stripped from the liquid film through the action of aerodynamic forces. Previously, efforts were made to model aerated-liquid injection [1] using a volume-of-fluid (VOF) method. This method accurately captures the interfaces between the large bubbles prior to their passage through the nozzle. However, as the two phase flow enters the nozzle, the VOF method loses its effectiveness due to the combination of high flow speeds and inadequate resolution of surface tension effects. The result is a ‘mixed out’ flow in which individual droplets cannot easily be discerned.

![Figure 9.1: Centerplane snapshot of density contours inside an 'out-in' aerated liquid injector](image)

The centerplane snapshot of density from the simulation of flow within an ‘out-in’ aerated-liquid injector configuration is shown in Figure 9.1. The mesh renders both the interior of the injector and part of the region outside the injector using upwards of 80 million cells. Aerating gas enters the center mixing nozzle through an array of small ports. The figure clearly shows the merging of large bubbles in the mixing chamber along with the breakdown of the structure into a highly turbulent, two-phase flow within the nozzle. The two-phase mixture further expands outside the nozzle, resulting in a turbulent, two-phase jet with significant liquid content, as indicated by the vapor-phase mass fraction contours of Figure 9.2. The initial rapid expansion of the plume should be particularly noted. Since the nozzle exit pressure is much higher than the ambient pressure, there is a free-jet expansion that terminates with a Mach disk. The mixture
model predicts supersonic flow in this region by virtue of its use of a mixture sound speed, which reaches values less than that of either the gas or the liquid for volume fractions between $\sim 0.2$ and $\sim 0.8$. By this point, secondary atomization processes have created a large population of droplets, and the assumption of negligible velocity slip implied in the two-phase mixture model is no longer valid. This sets the stage for the use of the droplet-tracking module to predict the spray evolution.

The initial conditions for the droplet field, in terms of the number and distribution of the droplets, were determined by first evaluating the liquid-phase mass within a small volume at the end of the nozzle. Using the core-annular assumption mentioned previously, the injecting mass is partitioned into two regions: an annular region assumed to be composed of 50-micron droplets and a core region encompassing the remainder of the exit plane. Assuming a droplet size for the core regions enables the determination of the population of droplets within each region at every time step. The frequency of droplet injection is adjusted to ensure that the mass flow rate of the injecting droplets closely matches that of the continuous liquid phase. The results shown next assume a one-way coupling in which the two-phase flow acts as a carrier for the droplet population, but is not influenced by it.

For the calculations, two core droplet size of 10$\mu$m and 20$\mu$m have been assumed. The population of droplets within the domain reaches a (statistically) stationary state after one flow-through time. The assumed core droplet size determines the size of the population. A smaller value results in an average population of $\sim 7.5M$ droplets while a larger droplet size leads to an average population of $\sim 0.8M$ droplets. Figure 9.3 shows extractions of the droplet field near the centerplane (+/−2mm) for both populations. Note that in case of the 10$\mu$m
droplets, the figure only shows every tenth droplet for legibility. Droplets are scaled by their relative size. The larger outer droplets from the annulus remain in the outer parts of the plume. Owing to their larger size, they retain much of their outward direction imparted during the initial expansion of the jet. The smaller core droplets tend to move outwards initially following the expansion process; however most of them quickly relax to the flow and turn towards the centerline after the Mach disk. Note that even among the smaller core droplets, the $10 \mu m$ droplets show minimum dispersion since they adhere to the flow strongly due to their smaller size and thus smaller Stokes number.

![Figure 9.3: Center plane droplet populations with 50 $\mu m$ annular droplets (shown in red) and 10 / 20 $\mu m$ core droplets (shown in blue)](image)

Figure 9.4 shows a composite image that super-imposes the vapor mass fraction contours as predicted by the mixture model with a snapshot of the $20\mu m$ core droplet population. Velocity slip effects are pronounced for the larger annular droplets as well as smaller droplets located near nozzle walls. As seen from the figure, the spreading rate of the droplet field is much larger than that of the gas field and shows qualitative agreement with recent X-ray fluorescence measurements [13]. It is important to note however that two-way effects have not been included in these results.

Figure 9.5 compares the plume structure with experimental shadowgraph imaging [12]. In these images, the droplet field is not restricted to the centerline, the droplet scale is uniform, and five time-sequenced populations are shown. It is again obvious that the $10\mu m$ population remains closer to the centerline. The predicted spreading rate is very close to that observed in the experiment.
Figure 9.4: Composite image super-imposing vapor-phase mass fraction contours with a droplet field snapshot

Figure 9.5: Comparison of predicted droplet fields with experimental shadowgraph results
9.2 Droplet Transport in a C-17

The second case presented in this thesis showcases the droplet tracking engine’s capacity for handling immersed boundaries. The simulation relates to modelling contaminant transport in an internal flow in a C-17 aircraft equipped for medical evacuation. Each feature in the cabin (people, seats, beds) is rendered as an immersed boundary as shown in Figure 9.6. The configuration contains upwards of 3000 mesh blocks (with a total of 62.5 M cells) and is mapped to 512 processors.

![Figure 9.6: Rendered interior of C-17 MedEvac showing mesh slices (see closeup view for detail)](image)

In the simulations discussed in this section, one person within the cabin sneezes. This sneeze is simulated using a population of droplets. The droplet population is assumed to contain 5000 droplets, with uniformly distributed diameters over a range 5 µm to 100 µm. All droplets are injected with an initial velocity of 3.5 m/s. The direction of initial droplet motion is associated with a normal vector that extends outward from the subject’s breathing area. Five simulations were performed, with five out the six patients that lie on the beds being sources for droplets. Two of these patients are located on the lowest bunk while the remaining three patients are on the top bunk of the litter arrangement. Droplet transport is driven by the local velocity field. This velocity field is dictated by a combination of the internal HVAC and vent systems, the breathing
and thermal plume effects due to the people inside the aircraft. Note that droplet transport is also affected by gravitational settling. If the droplet crosses a physical or an immersed boundary then the motion of the droplet is halted, as discussed in Chapter 6, to mimic deposition. Simulations were conducted for \( \sim 20 \) minutes of physical time. Figure 9.7 shows a side view of re-suspended droplets for each case, extracted 125 s after the sneezing event. The droplets are colored based on their size, with the smallest ones being blue and the larger ones being red. The larger droplets immediately fall back on the patient emitting them. However, the smaller droplets do remain aerosolized. The emission event is highly influenced by the patient’s location - small droplets emitted by persons lying on the topmost bunks rise quickly and are rapidly entrained into the cabin circulation loop. Droplets emitted by persons lying on the lower bunks are influenced more by the complex flow patterns found in the vicinity of the objects - this leads to lateral as well as vertical dispersion.

As time passes, all the aerosolizable droplets get entrained into the main cabin’s circulation that is driven by the vent system. These droplets thus get transmitted throughout the cabin. Most of them eventually deposit on either a physical surface or an immersed boundary after 20 minutes of physical time. Figure 9.8 shows the droplet field corresponding to Case 1 at different time steps. The source is toward the rear end of the cabin for this case. Also note that many droplets are deposited on the cargo ramp.
Figure 9.7: Aerosolized droplet population at 125 s after sneeze for different patients
Figure 9.8: Droplet locations at different times (Case 1)

Figure 9.9: Deposition patterns for different sneezing events after 20 min
The potential exposure of various persons in the cabin and possible exposure pathways for each individual event can be examined by counting the number of aerosolized droplets that deposit on the other people within the C-17. The outcome of this analysis can be seen in Figure 9.9. Every large dot represents the position of the individuals in the aircraft in $X - Z$ space. Note that the front of the cabin is towards the top. The colors of the dots represent the number of aerosolized droplets deposited on that person, with red being the largest and blue being zero. Since relatively few droplets ( ~ 500 to 700 out of 5000) are actually aerosolized, the source is exposed the most. The cabin circulation inside the plane results in some deposition events on individuals relatively far away from the source. Sources located nearer to the front of the cabin (Cases 3 and 5) result in more local exposure. Figure 9.10 illustrates one reason for this phenomenon for Case 5. Droplets initially moving upward can get captured in the low momentum region upstream of the wing-root housing and can then deposit preferentially in the front part of the cabin.
9.3 Air Assisted Liquid Drop Atomization

Air assisted atomization of a single liquid droplet is simulated using the developed droplet engine, as a means of validating the breakup methodology. The results are then compared to the experimental results presented by A.B Liu and R.D. Reitz [16]. In this experiment, a stream of monodisperse droplets is injected into an air jet flowing in the transverse direction to the droplets. Due to the air jet, droplets break up and the trajectories turn towards the direction of the impinging air jet thus creating a plume. An illustration of the experimental setup is shown in Figure 9.11.

![Figure 9.11: Schematic diagram of the experimental apparatus](image)

As seen from Figure 9.11, air is injected vertically in the domain using a contoured nozzle with a diameter \(D\) equal to 9.525 mm. The converging section has an \(R/D\) ratio of 0.5, where \(R\) denotes the radius of curvature. This design of the injector is chosen to ensure a flat velocity profile at the nozzle exit. The experiment presents results for air jet velocities ranging from 59 m/s to 250 m/s. This allows the observation of the droplet breakup phenomenon for Weber numbers ranging from 18 to 319. The injected droplets have a diameter of 170 \(\mu m\) and are injected every 25 \(\mu s\) with a radial velocity (in the +X direction as seen in Figure 9.11) of 16 m/s. As these droplets interact with the air jet, their trajectories change. Aerodynamic drag results in droplet breakup, which creates smaller droplets which respond to the incoming air jet. The liquid used in these experiments is Benz’s Oil UCF-1 test fuel with density \(\rho = 824kg/m^3\), dynamic viscosity \(\mu = 2.17^{-3}Pa.s\) and surface tension coefficient \(\sigma = 0.02kg/s^2\). The droplets
are injected 2mm away from the nozzle exit and 25mm away from the central axis of the nozzle.

The flow in the domain is solved using a multi-block, two dimensional grid with \( \sim 22,000 \) elements, with the assumption that the flow is axisymmetric. The REACTMB CFD solver was used to perform a RANS simulation of the flow using the Menter \( k - \omega \text{SST} \) turbulence model. Results are presented for the case where Weber number is equal to 18, and the velocity at the nozzle exit is 59\( m/s \). The flow in the domain is shown in Figure 9.12. A comparison of the experimentally measured velocity profile and the computed profile 2\( mm \) from the nozzle exit is shown in Figure 9.13. Note that this velocity profile is plotted with data extracted at the cross section of the two dimensional flow field at the dashed line shown in Figure 9.12.

![Figure 9.12: 2D velocity profile for \( We = 18 \)](image)

As seen in Figure 9.13, the computed velocity profile shows good agreement with the experimental measurements. The computed velocity field was then mapped to a three dimensional (\( 101 \times 101 \times 41 \)) Cartesian grid, and only the droplet tracking engine was run to simulate the droplet trajectories. The simulation was run for 0.25 \( s \) to inject 10,000 droplets. The constant Weber number approach in which a separate \( t_{bu,r} \) is calculated for the radius factor, is used

71
for this simulation. However, all approaches to calculating $t_{bu}$ give similar results, since the radius factors are very close to 0.79; thus every breakup event creates two new droplets. Since the timing of the breakup event is controlled by the TAB model, the location of breakup remains exactly the same for all cases. Figure 9.14 shows a snapshot during this simulation. The scatter data (denoted by the black circles) represents the experimental observations for droplet trajectories.

![Figure 9.14: Comparison of Droplet trajectories](image)

As seen from Figure 9.14, the droplet trajectories are in reasonable agreement with the experimental data. However, the simulated trajectories overshoot slightly, not turning as much as the observed results. It is clear that overall trajectory is governed by not just the drag formulation used, but also due to breakup events. The exact location of breakup events significantly changes the trajectory; with each successive breakup, smaller droplets are formed which leads to a reduction in the Stokes number and the droplets responding more strongly to the air jet. As seen, there is a secondary trail of droplets which follows the experimental data more closely. The difference in trajectories may also be due to the fact that in reality, distorting droplets may have a higher drag coefficient, since they are not spherical. This would lead to higher
drag coefficients which in turn would cause the droplets to follow the flow more tightly. The
dispersion which causes the plume as seen in the right side of Figure 9.14 is caused due to the
addition of normal velocity components to product droplets after breakup. Additionally, due to
the use of stochastic droplet tracking, and the presence of significant turbulence in the shear
layer also introduces dispersion.

The Liu-Reitz experimental data also provides a distribution of the Sauter mean diameter
(SMD) with respect to radius at two locations - 29 and 47 mm away from the nozzle exit. A
comparison of the Sauter mean diameter distribution from the computed trajectories with the
experimental data is shown in Figure 9.15. As seen, the experimental results show a wider range
of product droplets, while the simulated droplet size distribution is bimodal. However, at the
core trajectory location, there is good agreement between the measured and predicted SMD.

![Figure 9.15: Comparison of computed Sauter mean diameter distribution with experimental results](image)

Figure 9.15: Comparison of computed Sauter mean diameter distribution with experimental results

Figure 9.16 shows a closeup view of the droplet trajectories emphasizing the locations of
droplet breakup events. The underlying flow field shows the width of the jet. As seen from the
figure the droplets breakup sequentially in a relatively small section of the jet and then simply
advect based on the underlying flow. The CAB model implementation assumes that radii of
all product droplets are equal; there is no sampling or randomization which would allow for a
distribution of droplets to be generated. As seen from the figure, we essentially get a sequential
binary breakup of droplets. Successive binary breakup occurs four times for most droplets;
thus one 170 µm droplet breaks up into $2^4 = 16$ droplets, to give a SMD value of 67.46 µm
exactly matching the simulation results. Additionally, some of these 67.46 µm droplets undergo
another binary breakup step, yielding droplets with size 53.54 µm. These two distinct droplet
sizes create two separate streams as seen in Figure 9.14, which thus results in a bimodal SMD
distribution downstream. The binary breakup is due to the the radius factor values being $\sim 0.79$
for the Weber numbers experienced by the breaking droplets. In essence a complete variation of droplet sizes observed in the experiment is not captured by the breakup model. However the core trajectory and the SMD observed in this trajectory are captured by the CAB model.

Figure 9.16: Close up view of the breakup process
9.4 Planar aerated liquid jet injection with droplet breakup

This case is a simplified version of that presented in Section 9.1, with a planar nozzle used and periodic boundary conditions imposed in the spanwise direction. The initial conditions for velocity, pressure and vapor phase mass fraction were obtained by time and space averaging the nozzle exit flow properties of the aerated liquid injector discussed in Section 9.1. The injection pressure is 750 KPa, injection velocity is 65 m/s and the gas mass fraction is 0.04. The emphasis in this section is on evaluating the performance of the droplet breakup model. An air-water mixture is injected at high pressure from a jet with a rectangular cross section into a channel at quiescent conditions. The high pressure jet initially expands radially outwards and then is turned inwards by the formation of a Mach disk. Since the underlying flow is supersonic, high relative velocities are observed thereby causing a significant number of new droplets due to multiple breakup events.

Figure 9.17: U-velocity component distribution at Z = 0

The flow in the domain is simulated by the REACTMB CFD solver with turbulence modelled using LES. The domain is split into 152 blocks mapped to 152 processors. The simulation was
run for 2500 iterations and 250,000 droplets were injected into the domain over this time. The injected droplet population is monodisperse with a droplet diameter of 10µm, with 1000 droplets being injected every 10 iterations. The injected droplets were assigned velocities equal to the interpolated flow velocities at the droplet location. The droplet breakup process is modelled with the CAB model as discussed in Chapter 4. The distribution of the U-velocity component at the $Z = 0$ cross-section of the domain is shown in Figure 9.17.

Figure 9.18: U-velocity component distribution at $Z = 0$

Figure 9.18 shows an isometric view of the droplet population at the end of the simulation. A wide range of droplet sizes is observed in this case. Overall, while 250,000 droplets are injected, the total droplet population at the end of the simulation is $\sim 4M$. The evolution of the droplet population from injection over the course of the simulation is shown in Figures 9.19 - 9.23.
Figure 9.19: Droplet population after 250 timesteps

Figure 9.20: Droplet population after 750 timesteps
Figure 9.21: Droplet population after 1250 timesteps

Figure 9.22: Droplet population after 2000 timesteps
As seen from Figures 9.19 - 9.23, the injected droplets follow the flow as it expands radially outwards. Most of the droplets in the interior of the jet turn back inwards, following the flow due to their small size. However, droplets on the periphery of the jet undergo stripping breakup due to the large velocity gradient between the high speed jet and the quiescent surroundings. These smaller droplets stay in the free jet boundary and follow the flow even more tightly. Droplets crossing the Mach disk undergo catastrophic breakup, resulting in a sharp reduction in droplet size followed by a corresponding increase in the number of droplets. These droplets get entrained in the low speed region behind the shock, while the surrounding larger droplets move around the Mach disk. Overall, the flow is highly transient and the smaller droplets generated follow the flow very closely. These droplets very clearly trace the vortical structures and eddies in the flow.

Note that the frozen $K_{br}$ based $t_{ba}$ interpretation was also used to simulate this case, with the same underlying flow. That resulted in the generation of $\sim 4.3M$ droplets, thus higher than those generated with the $t_{ba,r}$ but fairly close.
A closeup view of the velocity profile at shock and the droplet interaction with the shock is shown in Figure 9.24. The plot on the left shows the location of the shock and the velocity field around it. As expected, there is generation of a large number of droplets. This response is successfully captured by the droplet breakup model. The flow around the shock carries the larger droplets forward. Here too, the droplets which encounter the free jet boundary break up and get entrained in the turbulent eddies formed in the shear layer.

Figure 9.24: Droplet breakup due to shock
9.5 Injection into a Subsonic Cross-flow

The details of the case presented in the first part of the section have been described in [26]. The results presented here correspond to an experiment described in [11] involving normal injection of water at a flow rate 2.3g/s and a gas-to-liquid mass ratio (GLR) of 4.45% into a cross-flowing air stream at Mach 0.3, 298 °K, and atmospheric pressure. For calculations the GLR used was 4%, the difference being the inclusion of a dopant (Krypton gas) in the experiment to facilitate confocal X-ray fluorescence imaging. An aerated liquid injector with nozzle exit diameter \( D = 0.5\, \text{mm} \), and a nozzle length-to-diameter ratio of 10, was used for injection. Under these conditions, the flow inside the nozzle is highly unsteady, transitioning from a slug-type two-phase flow to a core-annular two phase flow. Figure 9.25 shows the snapshots of instantaneous liquid density and time averaged liquid density.

\[ \Omega_{liq} = \frac{\dot{m}_{liq} \Delta t}{\rho_p} \]  

(9.1)

In case of a core-annular type droplet injection, the annular droplet size is assumed to be 20 \( \mu \text{m} \), while the core droplets are set to 10 \( \mu \text{m} \). Details about injection of this type have been
described in Chapter 8. The log normal initialization is facilitated by first sampling a log-normal size distribution with geometric mean of $7.5 \, \mu m$ and a standard deviation (in the logarithmic coordinate) of 0.5. After each sampling event, the diameter is stored and the droplet volumes summed until the target volume is reached. This population of droplets is also distributed randomly over the entrance plane as seen in Figure 9.26.

The mass flow rate of the liquid phase extracted from the continuum mixture flow and from the Lagrangian droplet population sampled at $X/D = 25$ (12.5 mm from the injector) is plotted versus time in Figure 9.27. The droplet population is sampled by binning droplet data within 1 mm x 0.5 mm x 0.5 mm volumes oriented within the plane in question. The mass flow rate through the $n^{th}$ volume at a given time instance is defined as:

$$\dot{m}_n = \frac{1}{\Delta x} \rho_p \sum_{k=1}^{N_p} \frac{\pi}{6} d_{p,k}^3 u_{p,k}$$

(9.2)

The total mass flow rate is found by summing over the number of volumes that comprise the plane in question.

![Injectant entrance plane droplet populations](image.jpg)

**Figure 9.26:** Injectant entrance plane droplet populations

As seen in Figure 9.27, there is a large spike in the liquid mass flow rate, corresponding to the passage of a slug of liquid water through the exit plane of the nozzle. The average mass flow rate over the time interval of $\sim 0.006 \, s$ is about 2.63 \, g/s which is somewhat larger than the target experimental value of 2.3 \, g/s.
9.5.1 Results without Droplet Breakup

Figure 9.28 shows the droplet populations colored and sized according to the droplet diameters, for both the core-annular and log normal initial droplet distributions. The X location at which cross-flow droplet distributions are extracted is represented by the vertical bars on the left of the image. These distributions are extracted at $t = 0.00165\, s$, when there are $\sim 1.5 \, M$ droplets in the domain. This number increases to $8.8 \, M$ through the course of the simulation. The log normal distribution generates a broader plume; this is due to the fact that a wider range of droplet sizes...
form the distribution. Different droplet sizes respond differently to the flow, creating a broader plume. The core-annular bi-modal distribution leads to the 10 $\mu$m core droplets preferentially remaining near the centerplane and near the edge of the shear layer. Due to their low Stokes number, these droplets follow the flow tightly. Some droplets are entrained in counter-rotating vortex pairs and are forced toward the wall. The 20 $\mu$m annular droplets are distributed more broadly.

Confocal X-ray fluorescence data is available for the gas-phase injectant (nitrogen doped with krypton gas) as well as the liquid spray (water doped with bromine). Figure 9.29 compares predicted gas-phase density contours with experimental data. A Mach disk is initially formed upon expansion into the flow – this feature is not shown in the experimental results. The predicted vapor plume is broader and does not penetrate as far as the experimentally-measured plume. The absence of two-way coupling may influence the degree of predictive capability, as the spray is very dense (liquid mass fractions in excess of 95%) in this region.

Figure 9.29: Comparison of injectant gas density predictions with confocal X-ray fluorescence measurements

Figure 9.30: Liquid density predictions versus confocal X-ray fluorescence measurements
The time averaged droplet properties shown in these figures are calculated by averaging instantaneous properties for each volume. The instantaneous velocity (as an example) for the \( n^{th} \) volume is determined by:

\[
v_{p,n} = \frac{\sum_{k=1}^{N_{p,n}} \frac{\pi d_{p,k}^3}{6} v_{p,k}}{\sum_{k=1}^{N_{p,k}} \frac{\pi d_{p,k}^3}{6}}
\] (9.3)

These values are then time-averaged over the plane in question.

Figure 9.30 shows the comparison of the predicted liquid densities (mass of liquid per total volume) with the confocal X-ray measurements, along the centerplane. The spray is less dense in its core in the experiment. This can be explained by a lateral expansion of the spray initially, resulting in dispersion of the liquid. The core-annular initialization leads to less liquid dispersion than the log-normal initialization.

Figure 9.31 shows a comparison of the time-averaged droplet axial velocity predictions at the centerplane and at \( X/D = 25 \) with PDPA data. As seen, the droplet velocity field is predicted much better using the log-normal initial distribution. When the core-annular assumption is used, droplets tend to concentrate nearer to the surface. Also, their distribution follows the counter-rotating vortex pairs that originate due to the normal injection event. For the log-normal distribution, we get an elongated plume along the vertical direction; however, it is not

Figure 9.31: Comparison of axial droplet velocity predictions with PDPA data
as elongated as seen from the PDPA results.

![Figure 9.32: Comparison of centerline Sauter mean diameter predictions with PDPA data](image)

The centerline Sauter mean diameter predictions are shown in Figure 9.32. Again, the results with a log normal initial droplet distribution are superior to the core-annular distribution results.

### 9.5.2 Results with Droplet Breakup

Droplet breakup was activated for the log-normal case midway through the simulations to see the response that the breakup models would have on the distribution of droplets. Simulations were then continued for a maximum duration of $0.0024 \text{s}$ (about 3.6 times the flow through time based on a flow velocity of $100 \text{ m/s}$ and a length of $0.066 \text{ m}$). The droplet distribution when breakup was turned on is as shown in Figure 9.33. Note that the snapshot of droplet population shown in Figure 9.33 corresponds to a time instant near the peak of liquid injection. Results are presented for two different interpretations of the breakup model, as well as with no breakup. At $t = 0.0 \text{s}$, there are $8 \times 10^6$ droplets in the domain. Note that the droplet distributions show every $25^{th}$ droplet actually present in the domain. The droplets shown in these images are colored and sized by their diameter, and the images correspond to $t = 0.9 \text{ ms}$ after the initiation of the simulations.
In the case with no droplet breakup as seen in Figure 9.34, there is a slight elongation of the plume in the axial direction. This is due to the fact that the droplet population at $t = 0$ shown...
in Figure 9.33 corresponds to the peak of the liquid injection mass flow rate, resulting in a more vertical injection. As the flow rate drops over the course of the simulation there is slightly lower penetration, and the plume elongates due to the axial air flow. After 0.9 ms, there are a total of 5.4M droplets in the domain. The reduction in the number of droplets is due to the reduction of liquid flow rate into the domain and droplets naturally flowing out of the domain.

Figure 9.35 shows the droplet population after 0.9 ms using the constant Weber number based $t_{bu,r}$ for calculating the radius factor. The total number of droplets in the domain at $t = 0.9$ ms is 18.6M. As expected after droplet breakup, there is a denser population of smaller droplets. The smaller droplets naturally result in further reduction in penetration rate, since smaller droplets have a smaller Stokes number and thus relax to the axial flow more quickly.

The droplet population using $t_{bu}$ calculated by the frozen $K_{br}$ approach is shown in Figure 9.36. This approach generates a huge number of droplets - the domain has 105.8M droplets after 0.9 ms. This approach thus produces a significantly larger number of product droplets as compared to the constant Weber number based $t_{bu,r}$ method. This can be clearly seen by the presence of a dense blue region throughout the domain. This response is due to the fact that smaller droplets which break later in the simulation have a large $t_{bu}$ value, and thus have a lower radius factor value at breakup. This generates a lot more droplets. However, this does not seem to be a physically consistent response - a majority of breakup events should occur as the injected flow turns, since the Weber number in those regions will be higher. As the larger droplets break up, they produce smaller product droplets, which should relax to the flow
and thus the number of breakup events further downstream must be much lower. Due to the transient nature of the flow, some breakup events might still occur, but the distribution and density of the smaller droplets in Figure 9.36 points to a somewhat erratic breakup response. In case of Figure 9.35 on the other hand, the density of the smaller droplets visibly dissipates as we move downstream. Note that the integral formulation for calculating $K_{br}$ was tried as well - that approach produced over a billion droplets and the processors ran out of memory.

Figure 9.36: Droplet distribution at $t = 0.9\text{ms}$ using $t_{bu}$ with frozen $K_{br}$

Figure 9.37: Closeup view of droplet generation due to different approaches for calculating radius factor
A closeup view of droplet population captures the difference in droplet generation due to the radius factor calculation approaches, based on $t_{bu}$ (with a frozen $K_{br}$) and $t_{bu,r}$. As seen in Figure 9.37, using a $t_{bu}$ based radius factor results in the formation of streaks of tiny droplets, due to a low radius factor value. However, in the case of the radius factor calculated with $t_{bu,r}$, the droplets have reasonable sizes. There are no clumps and droplets are well dispersed, indicating that the $t_{bu,r}$ based radius factor definition may be the better approach to model droplet generation.

For the simulation results presented above, the evolution of certain droplet quantities averaged over the entire domain has been plotted to understand the global characteristics of the droplet population. Figure 9.38 shows the evolution of the Sauter mean diameter ($d_{32}$) and the arithmetic mean diameter ($\bar{d}$) of the droplets in the domain.

\[
d_{32} = \frac{\sum_{i=1}^{N(t)} d_i^3}{\sum_{i=1}^{N(t)} d_i^2} ; \quad \bar{d} = \frac{\sum_{i=1}^{N(t)} d_i}{N(t)} \quad (9.4)
\]

Here, $N(t)$ represents the total number of droplets in the domain at time $t$. Similarly, Figure 9.39 shows the evolution of the volume averaged axial and vertical velocities. Volume averaged quantities were defined as follows:

\[
\phi = \frac{\sum_{i=1}^{N(t)} \phi_i d_i^3}{\sum_{i=1}^{N(t)} d_i^3} \quad (9.5)
\]
As seen from Figure 9.38, the target arithmetic mean of $\sim 7.5 \, \mu m$ (imposed through the log-normal inflow droplet distribution) is maintained in the no-breakup case. The Sauter mean diameter values are higher, as expected. The average droplet sizes reduce significantly once droplet breakup is initiated for both $t_{bu}$ and $t_{bu,r}$ based definitions for calculating radius factor. The $t_{bu}$ based approach produces a large number of very small (micron sized) droplets. The simulation with the $t_{bu}$ approach thus had to be curtailed due to rapid degradation in performance.
as some processors were near their core limits. This shows the importance of including dynamic load balancing in future efforts. The $t_{bu,r}$ based approach to generating new droplets shows an initial decrease in average droplet sizes followed by recovery - however no steady state was reached during the time allotted for the simulation. Figure 9.39 shows that the smaller droplets produced as a result of breakup tend to follow the axial flow more strongly. As a result, there is an increase in axial momentum but a loss of vertical momentum.

The following images show the distribution droplet properties extracted at a cross-section at $X = 50\,mm$. The distribution of Sauter mean diameter as well as the axial and vertical velocity components are compared with the experimental results. The constant Weber number based $t_{bu,r}$ has been used to calculate radius factors in this simulation. Note that the computed values were time averaged over a simulation time of 1.2 ms.

![Figure 9.40: Comparison of Sauter mean diameter distribution](image1.png)

![Figure 9.41: Comparison of U-velocity distribution](image2.png)
As seen from Figures 9.40 - 9.42, the use of the breakup model does not improve agreement with the experimental results. This can be attributed to the fact that the assumed log-normal distribution of the droplets already has a smaller mean value, as compared to the experiment. The results are consistent in that smaller droplets generated closely follow the flow. These droplets generally remain within the counter-rotating vortex pairs that originate upon injection of the liquid.
9.6 Single Droplet Vaporization

Droplet vaporization of a single, isolated hexane droplet suspended in a constant velocity stream of air was simulated using the droplet engine. Simulated results are compared to the experimental results presented in Miller et. al. [18]. The droplet has an initial diameter of $1.76 \text{ mm}$, initial droplet Reynolds number of 110 and initial temperature equal to 291 °K. The ambient gas was air, with a constant temperature equal to 437 °K. Note that droplet advection equations were not solved, since the droplet is stationary (hanging from the end of a thin wire) in the experiment. The effect of the droplet on the flow has been neglected as well.

![Figure 9.43: Temporal evolution of $D^2$ and temperature for a single isolated Hexane droplet with $Re_{d,0} = 110$](image)

Note that $D$ in Figure 9.43 denoted the droplet diameter. As seen from the results presented,
the LK-1 model used shows good agreement with the experimental results. The droplet Reynolds number and the ambient temperature for the simulated case lead to moderate evaporation rates where non-equilibrium effects are present. These are captured well by the droplet vaporization model used as a part of the droplet tracking engine. Future work will couple the vaporization model with advection and breakup processes, thus leading to a complete module for simulation of the liquid fuel injection process.
Chapter 10

Conclusions

10.1 Discussion of Current Work

A Lagrangian droplet tracking engine for three dimensional, structured, curvilinear multi-block grids has been developed and tested for a variety of flow conditions. A linear time search algorithm has been developed to allow for fast search of the cell which contains the droplet. Flow variables are interpolated to the droplet location using trilinear interpolation to calculate the forces acting on the droplet. The droplet is advected by accounting for various forces such as drag, lift and gravity, and the droplet position is integrated using a semi-implicit scheme. Physical models to simulate droplet breakup and vaporization have been implemented in the engine in an effort to better simulate spray dynamics. Droplet breakup phenomenon is modeled using the Taylor Analogy Breakup (TAB) model. Multiple approaches to droplet generation have been considered based on the Cascade Analogy Breakup (CAB) model. Droplet vaporization has been modeled by evolving the droplet temperature and mass flow rate due to evaporation. Non equilibrium evaporation effects are included using the Langmuir-Knudsen law. The droplet engine can also handle immersed boundaries (IBs) in terms of droplet injection from marked IBs and droplet deposition on IBs in the domain. The developed engine is efficient enough to simulate tens of millions of droplets and is parallelized using OpenMPI using the same domain decomposition as the CFD code. To this end, a new grid connectivity structure has been established and pre-computed.

As seen from the results, the droplet engine can effectively capture slip velocity between the two phases and remains stable for large values of slip velocities. The droplet response to the underlying flow based on variation in droplet size is physically consistent. However, this response is currently restricted to one-way coupling between the droplets and the flow. Droplet interaction with immersed boundaries is also captured successfully. However any interaction is currently limited to deposition. Droplet breakup simulated with the combination of the TAB and
CAB models captures a very important physical phenomenon which is critical to successfully simulating spray combustion. As seen from the results, the exact interpretation of the CAB model for calculating radius factor is unclear. The use of a constant Weber number based $t_{bu,r}$ seems to show the most promise and reasonable agreement overall. A direct correlation between the radius factor calculated and the Weber number at breakup makes physical sense and droplet populations generated with this approach show good agreement with experimental data in certain aspects. The results for injection into subsonic cross-flow clearly show the need to better understand the droplet distributions close to the injector. Capturing droplet populations near the injector may require a better tuned droplet breakup model working in conjunction with a two-way coupled engine to truly resolve the distribution of both phases. As seen from the Liu-Reitz experimental results, in reality, single droplet atomization yields droplets with a wide range of sizes. While the core trajectories and Sauter mean diameters predicted by the model show reasonable agreement, additional work may be necessary to understand and tune the model response to accommodate generation of various sized product droplets sampled from distributions. The CAB model currently does respond to various droplet breakup regimes; however, it may be necessary to also understand the distributions of droplet sizes each regime generates. The current version of the code successfully handles tens of millions of droplets across a few thousand processors without loss of data, pointing to a good parallel implementation and decent efficiency of the overall code. However, as the droplet population sizes continue to increase, penalties arising from the lack of load balancing start to become significant. This problem is exacerbated by droplet breakup, as localized breakup events can lead to a sudden rise in the number of droplets tracked by a single processor. Finally, droplet vaporization has been tested and validated for a single droplet. While the results show agreement with experimental data, further testing is required to ensure the model can effectively simulate vaporization in high speed flows in conjunction with advection and breakup processes. Additionally, the current implementation of the vaporization process is also one-way coupled; as droplets evaporate the mass of the vapor is not accounted in the continuum flow equations.

10.2 Future Work

Given the documented ability of the droplet engine to simulate a very large number of droplets, implementation of dynamic load balancing is the most critical extension possible to the current work. This can allow for better resolution of sprays and better tuning of the droplet breakup model. It will also make the overall droplet engine extremely cost effective, which may be necessary to incorporate two way coupling effects. For dispersed sprays with a low mass fraction of the discrete phase, a one-way coupled engine can provide good results. However as seen from the subsonic injection case, to truly capture the complexity of high pressure sprays, an extension
of the current CFD code so that the gas phase responds to droplets through source terms may be necessary. Effectively simulating droplet vaporization will also require two-way coupling between the CFD solver and the droplet engine with source terms for evaporated droplet mass and for energy transfer between the phases. Thus, dynamic load balancing and two-way coupling between the droplet engine and the CFD solver are two key avenues of future development.

Besides these two aspects, modelling droplet collisions will also lead to a significantly better resolution of dense sprays. However, efficiently modelling droplet collision may necessitate higher levels of abstraction with better data structures to maintain relational data between droplets. Additionally, physics based models for droplet interaction with walls may also be added; these may involve droplet deposition or splash-back. Similarly, accounting for turbulent re-suspension of deposited content back into the flow could also be a possible direction for further development.
REFERENCES


[15] Kuo-Cheng Lin, Christopher Rajnicek, Jonathan McCall, Benjamin Fischer, Campbell Carter, and Kamel Fezzaa. Structures of aerated-liquid jets injected from various nozzle...


